Liang-Jie (LJ) Zhang
Mario Jeckle (Eds.)

LNCS 3250

Web Services

European Conference, ECOWS 2004
Erfurt, Germany, September 2004
Proceedings

Springer

Liang-Jie (LJ) Zhang   Mario Jeckle (Eds.)

Web Services

European Conference, ECOWS 2004
Erfurt, Germany, September 27-30, 2004
Proceedings

Springer

Volume Editors

Liang-Jie (LJ) Zhang
IBM T. J. Watson Research Center, Business Informatics Department
1101 Kitchawan Road, Route 134
Yorktown Heights, NY 10598, USA
E-mail: zhanglj@us.ibm.com

Mario Jeckle (†)
Fachhochschule Furtwangen
Fachbereich Wirtschaftsinformatik
Robert-Gerwig-Platz 1
78120 Furtwangen, Germany

Library of Congress Control Number: 2004112843

CR Subject Classification (1998): H.4, H.3, H.2, H.5, C.2.4, I.2.11, K.4, K.6.5

ISSN 0302-9743
ISBN 3-540-23202-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 11325598 06/3142 5 4 3 2 1 0

Editor's Introduction

Welcome to the proceedings of the 2004 European Conference on Web Services (ECOWS 2004). ECOWS is one of the leading international conferences focusing on Web services. ECOWS 2004 was a forum for researchers and practitioners from academia and industry to exchange information regarding advances in the state of the art and practice of Web services, identify emerging research topics, and define the future directions of Web services computing. ECOWS 2004 had a special interest in papers that contribute to the convergence of Web services, Grid computing, e-business and autonomic computing, and papers that apply techniques from one area to another. This conference was called the International Conference on Web Services Europe in 2003. ECOWS 2004 was a sister event of the International Conference on Web Services 2004 (ICWS 2004), which attracted more than 250 registered participants in San Diego, USA.

Web services are characterized by network-based application components and a service-oriented architecture using standard interface description languages and uniform communication protocols. Industrial application domains for Web services include business-to-business integration, business process integration and management, content management, e-sourcing, composite Web services creation, design collaboration for computer engineering, multimedia communication, digital TV, and interactive Web solutions. Recently, Grid computing has also started to leverage Web services to define standard interfaces for business Grid services and generic reusable Grid resources.

The program of ECOWS 2004 featured a variety of papers on topics ranging from Web services and dynamic business process composition to Web services and process management, Web services discovery, Web services security, Web services-based applications for e-commerce, Web services-based Grid computing, and Web services solutions.

It is with great sorrow that we received the sad news of the sudden and untimely passing of Prof. Mario Jeckle, just before completing the paper review process. It was Mario who invited me to be the co-founder of ECOWS (aka ICWS-Europe) to help promote Web services research in Europe. I was very impressed by his enthusiasm. Together with all the organizing committee members and participants, we express our heartfelt condolences and our deepest sympathy to his family.

Finally, many people worked very hard to make the conference possible. I would like to thank all who helped to make ECOWS 2004 a success. The Program Committee members and referees all deserve credit for the excellent final program that resulted from the diligent review of the submissions. On behalf of the ECOWS 2004 Organizing Committee, I hope that all attendees enjoyed their stays in Erfurt and exchanged new ideas and explored the direction of Web services research and engineering.

Yorktown Heights, NY, USA

Dr. Liang-Jie (LJ) Zhang
July 2004

Organizing Committee

External Reviewers

Tobias Andersson
Morgan Ericsson
Marcus Edvinsson
Bing Li
Patrick Hung
Niklas Pettersson
Jiannan Wang
Tao Yu
Marc Lankhorst
Andries Stam
Peter Zoeteweij
Hugo W.L. ter Doest
Juan Guillen-Scholten
Henk Jonkers
Nikolay K. Diakov

Table of Contents

An Analysis of Web Services Workflow Patterns in Collaxa

Martin Vasko and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Wien, Austria
e0025379@stud3.tuwien.ac.at, dustdar@infosys.tuwien.ac.at

Abstract. Web services have a substantial impact on today's distributed software systems, especially on the way they are designed and composed. Specialization of different services is leading to a multitude of applications ultimately providing complex solutions. The interaction and modeling aspects of Web services is increasingly becoming important. Based on the needs for Web services conversations, process modeling, and composition, a variety of languages and technologies for Web services composition have evolved. This case study is focused on a systematic evaluation of the support for workflow patterns and their BPEL (Business Process Execution Language for Web Services) implementation in Collaxa, a leading BPEL process modeling and enactment engine for Web services processes.

Keywords: Workflow patterns, composition, BPEL

1 Introduction

The Business Process Execution Language for Web services (BPEL) [1] is an XML-based flow language that defines how business processes interact within or between organizations. The initial BPEL 1.0 specification was jointly proposed by IBM, Microsoft, BEA in August, 2002 and updated in May 2003 by version 1.1. It supports compensation-based business transactions as defined by the WS-Transaction specification. Business Processes specified in BPEL are fully portable between BPEL-compliant environments. BPEL is a block-structured programming language, allowing recursive blocks, but restricting definitions and declarations to the top level. The language defines *activities* as the basic building elements of a process definition. *Structured activities* prescribe the order in which a collection of activities take place. Ordinary sequential control between activities is provided by sequence, switch, and while. Concurrency and synchronization between activities is provided by the flow constructor. Nondeterministic choice based on external events is provided by the pick constructor. It contains handlers for events including message events (onMessage with portType, operation and partner) and timed events such as duration or deadline. Process instance-relevant data (containers) can be referred to in routing logic and expressions. BPEL defines a mechanism for catching and handling faults similar to common programming languages, like Java. One of the key aspects of Service Oriented Architectures is the support of dynamic finding and binding of services at runtime (e.g. in a repository such as UDDI). However, in BPEL the notion

of dynamic finding and binding is not supported directly. These activities need to be modeled explicitly (as activities, i.e. Web services).
Furthermore, BPEL allows describing relationships (third party declaration) how services interact (what they offer) by introducing *Partner*[1] *Link Types* (PLNK), with a collection of roles, where each role indicates a list of portTypes. At runtime, the BPEL runtime (execution) engine has to deal with the binding. BPEL also supports the notion of compensation and fault handling. Both concepts are based on the concept of *scopes* (i.e. units of compensation or fault). BPEL creates process instances implicitly, i.e. whenever instances receive a message, an instance is created. This is different to many workflow systems, which identify process instances by their ID. In the case of BPEL any "key field", such as an invoice number in an order fulfillment scenario, could be used for this purpose. The BPEL middleware has to deal with the issue of finding the suitable instance (or creating one if required). This mechanism is called *message correlation*.

The remainder of the paper is structured as follows. Section 2 introduces an example for a process model consisting of Web services. Section 3 analyses the Workflow patterns as suggested in [5] and the implementation found in Collaxa. Finally, section 4 provides remarks and summarizes the workflow pattern support.

2 A Supply Chain Process Model

The workflow patterns analyzed in this paper are discussed using a model of a Supply Chain process. The service consists of three different processes, each of them communicating with each other. The first process is a logistics process shown in Figure 1. It starts with an order placement as input, which can be generated through an interface for Web services. In the next step, the customer data is validated through a synchronous call to an external service such as a customer database. After user data processing, the decision between direct assembly and distributed processing has to be done. In case of direct assembly the shipment can follow as the next step. If the assembly has to be done before shipment, the process is stalled until all parts have arrived and have been compiled. Otherwise, the compilation can be done after shipment at the final destination. If this is the case, the service is stalled after shipment, until all components have arrived. The logistics control is responsible for the decision where the product has to be built. This brings us to the second process which is active in the Supply Chain process model.

The logistics control process receives input data from the logistics process. Based upon the total costs, calculated for the three different possibilities of assembly (before, after shipment or no direct assembly) it decides, which decision tree has to be followed. After execution of this sequence, the result of the best solution has to be found. It will be returned to the calling process and is from now on the headed execution target.

[1] was called Service Link Type in BPEL4WS 1.0

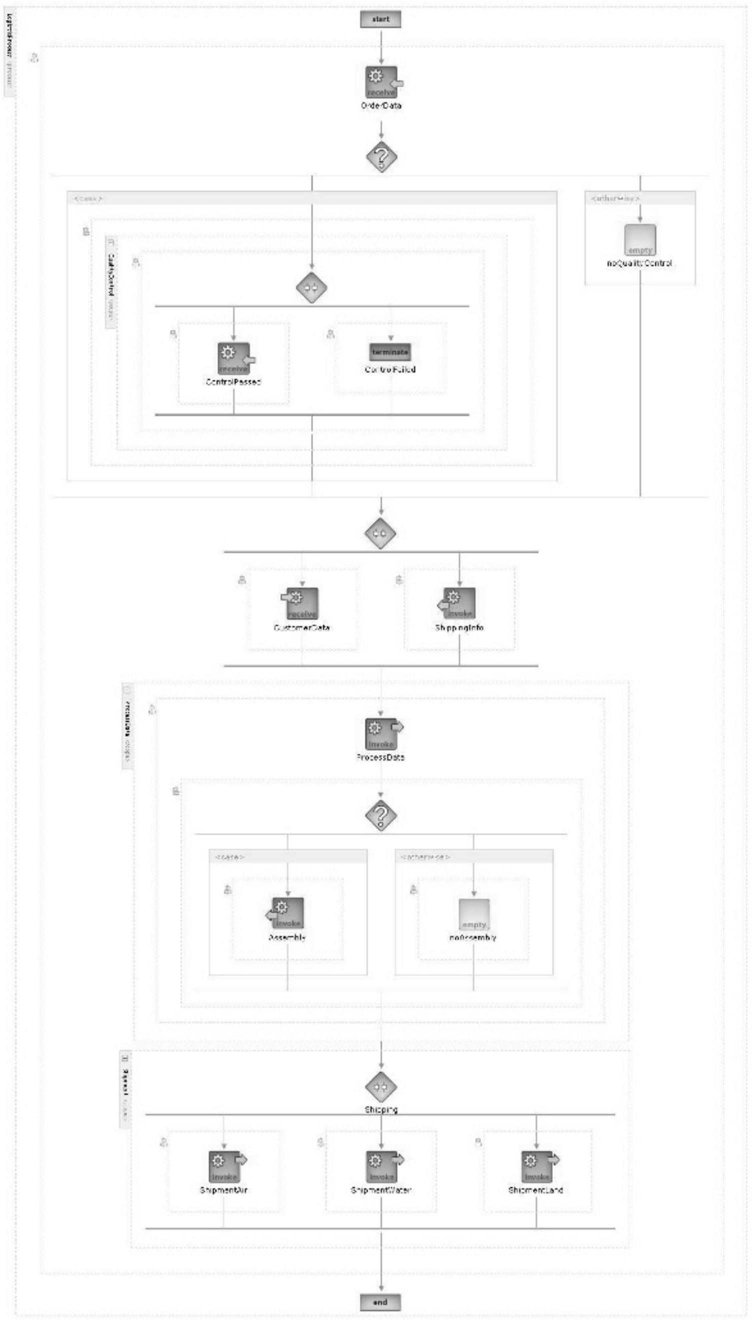

start
receive
OrderData
<case>
<otherwise>
empty
noQualityControl
QualityControl
receive
ControlPassed
terminate
ControlFailed
receive
CustomerData
invoke
ShippingInfo
invoke
ProcessData
<case>
<otherwise>
invoke
Assembly
empty
noAssembly
Shipping
invoke
ShipmentAir
invoke
ShipmentWater
invoke
ShipmentLand
end

Fig. 1.

3 Workflow Patterns

In the previous section an abstract view of a supply chain process was given. Now we take a closer look at workflow patterns that are constructed with basic BPEL elements. The pattern definitions were taken from van der Aalst et al. [5] in order to provide a framework for their systematic evaluation. Focusing on the support for these workflow patterns by Collaxa, we realised the majority of control structures in the logistics process model. More complex workflow patterns are presented in BPEL source code only. The provided figures visualize the explained patterns. To understand the structure of these figures the basic elements are described as follows: A service is referenced by a simple box. To visualize business flow, connected lines and arrows are used. The used notation is following the Activity Diagrams.

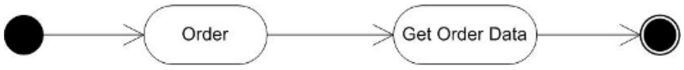


Fig. 2.

Workflow Pattern: Sequence

Two or more activities are processed in a workflow process in the order, in which they were defined. It does not support any kind of parallelism. The Collaxa BPEL engine realizes this with the concepts inherited from XLANG. Listing 1 provides a short excerpt from the source code.

```
Listing 1
<sequence name="main">
                 <receive name="receiveOrder" partnerLink="client"
portType="tns:SupplyChain" operation="initiate" variable="input"
createInstance="yes"/>
                 <receive partnerLink="client" portType="tns:SupplyChain"
name="CustomerData"/><scope>
</sequence>
```

In this example source code, there are two receive statements. The first gets the order data from an external Web service, and the second receives the customer data for verification.

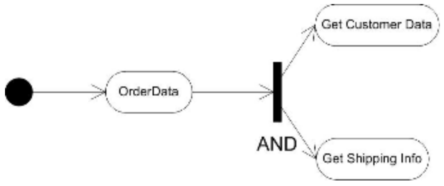


Fig. 3.

Workflow Pattern: Parallel Split

This pattern defines the structure of a process which is split into several threads of control, all executed in parallel. The order in which they are processed is not defined.

This pattern is provided by Collaxa by defining the flow activity, as described in the previous section. The source code which implements this structure is comparable to Listing 1 except, that the sequence statements are enclosed in a flow statement, as described in Listing 2.

```
Listing 2
<flow>
<sequence><receive partnerLink="client" portType="tns:SupplyChain"
name="CustomerData"/></sequence>
<sequence><invoke name="ShippingInfo" partnerLink="client"
portType="tns:SupplyChainCallback" inputVariable="output"/>
</sequence>
</flow>
```

The source code in Listing 2 contains the flow statement, which allows a parallel execution of the two sequences of control. The structure enclosed in the sequences is executed one after another.

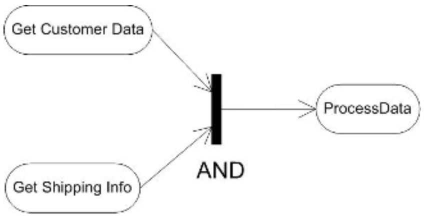


Fig. 4.

Workflow Pattern: Synchronization

This pattern is implemented mostly by the use of receive statements. The execution of the process continues when all data concerning the customer is collected. After that, the processing of the data will be started. In the actual example, the data for shipment and customers are collected. If both are finished, the 'normal' flow of the business process commences. This structure is shown in Listing 3. To show the synchronizing structure of this example, a scope is starting immediately after the flow statement, concerning the previous pattern to be executed.

```
Listing 3
<flow>
        <sequence><receive name="CustomerData"/></sequence>
        <sequence><invoke name="ShippingInfo"/></sequence>
</flow></sequence>
<scope name="ProcessData">
        <sequence><invoke name="ProcessData"/>
```

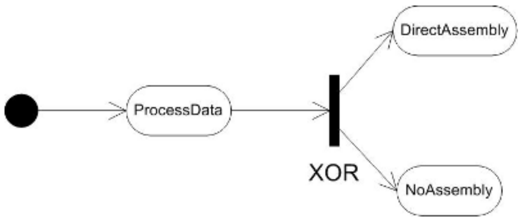


Fig. 5.

Workflow Pattern: Exclusive Choice

The exclusive choice structure defines a point in the business workflow, where a certain condition based on a decision in the flow is taken. This workflow pattern is best implemented with the switch statement of BPEL. In Listing 4 the execution of a switch statement is shown. This BPEL code fragment decides between 'direct assembly' or 'no assembly'.

```
Listing 4
<switch>
         <case>
         <sequence><invoke partnerLink="client"
portType="tns:SupplyChainCallback" name="Assembly"/></sequence>
         </case>
         <otherwise>
         <sequence><empty name="noAssembly"/></sequence>
         </otherwise>
</switch>
```

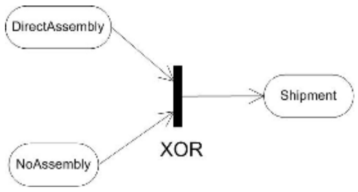


Fig. 6.

Workflow Pattern: Simple Merge

This pattern defines a point in the flow of execution, where two or more alternative branches come together. It is important to mention that the simple merge pattern does not support any kind of synchronization, which means, that none of the alternative processes is ever executed in parallel. This pattern is only supported in an indirect way by Collaxa.

```
Listing 5
<sequence name="assembly">
         <switch>
         <case><sequence><invoke partnerLink="client"
portType="tns:SupplyChainCallback" name="Assembly"/></sequence></case>
         <otherwise><sequence><empty name="noAssembly"/></sequence></otherwise>
         </switch>
</sequence>
<invoke name="Shipment"/></sequence>
```

In Listing 5 a simple merge pattern is provided. Notice, that this is not the only possibility to realize this functionality. The sequences contain one invoke statement and an empty statement. The decision is made by the logistics control, but in both cases, the alternative sequence is never chosen.

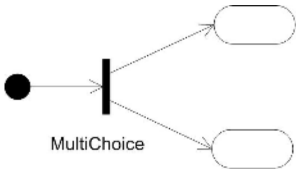


Fig. 7.

Workflow Pattern: Multi Choice

In contrast to the exclusive choice this pattern defines a point in the workflow, where a number of branches can be chosen. It is *not supported* by the Collaxa BPEL engine. Furthermore, it is rather complicated to achieve a similar pattern.

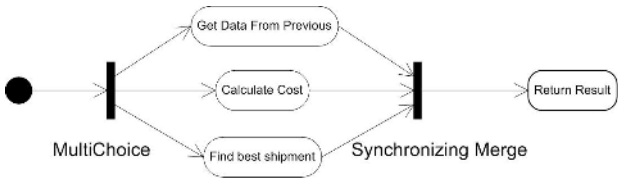


Fig. 8.

Workflow Pattern: Synchronizing Merge

This pattern marks a point in the process execution, where several branches merge into a single one. If one or more processes are active, the flow is triggered until these processes are finished.

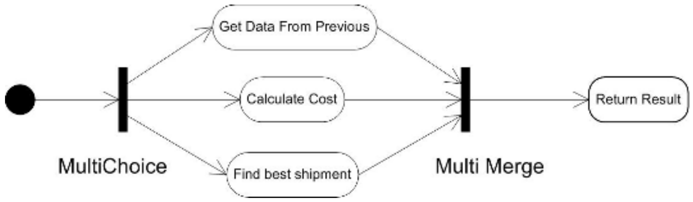


Fig. 9.

Workflow Pattern: Multi Merge

This pattern joins two or more different workflows without synchronization together. This means that results, processed on different paths, are passed to other activities in the order in which they are received. This pattern is neither supported by XLANG, nor by WSFL. Listing 6 provides a mixture between the two workflow patterns synchronizing - merge and multi – merge. The two sequences are always executed in parallel. The whole logic is enclosed by a loop statement. The pattern which is realized by this code depends on the problem to solve. It may have a synchronizing behavior, or alternatively, a multi merge one.

```
Listing 6
<switch>
<case><sequence>
        <flow>
                <sequence><invoke name="Get Data From Previous"/></sequence>
                <sequence><invoke name="Calculate Cost"/></sequence>
        </flow></sequence></case>
<otherwise><invoke name="Find best shipment"/></otherwise>
</switch>
<reply name="Return Result"/>
```

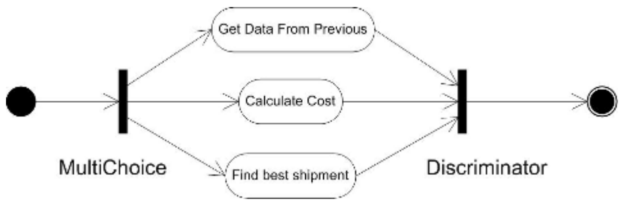


Fig. 10.

Workflow Pattern: Discriminator

This pattern describes a point in the execution flow of the system which waits for completion of an incoming branch, before executing a subsequent flow of control. As the subsequent process is activated, all other incoming branches are ignored. When all incoming branches are triggered, this structure resets itself to accept new incoming processes. This workflow pattern is comparable with the previous pattern. Note, that this structure is not supported by the BPEL language. Moreover, there does not exist a structured activity which can provide a workaround. A simple join condition in an OR condition is not suitable, because of the evaluation of the results. In this condition, always both or more results are evaluated before the execution path continues to the following activity. This restriction of BPEL makes it impossible for the Collaxa Engine to support this pattern, neither direct, nor as a workaround.

Workflow Pattern: Arbitrary Cycles

This pattern defines a point in the business process, where a portion of the process has to be “visited” repeatedly. There have to be no restrictions on the number, location, and nesting of these points. This pattern is not supported in BPEL. Although the while statement supports simple loop structures, it is not possible to jump into the loop flow in an arbitrary way. As in the previous pattern, the Collaxa engine does not support this pattern. This may be a possible extension for future BPEL specifications.

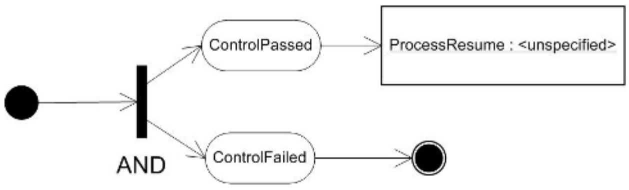


Fig. 11.

Workflow Pattern: Implicit Termination

An executed sub process is terminated, when there is nothing left to do. In BPEL the flow statement realizes this pattern. This activity awaits the occurrence of one set of events.

```
Listing 7
<scope name="QualityControl">
<sequence>
        <flow>
        <sequence><receive name="ControlPassed"/></sequence>
        <sequence><terminate name="ControlFailed"/></sequence>
        </flow>
</sequence>
</scope>
```

The sample code in Listing 7 demonstrates the QualityControl process. It is enclosed by a flow statement, which provides the parallel functionality. Depending on an external decision by a quality controller, the process flow can be approved or denied.

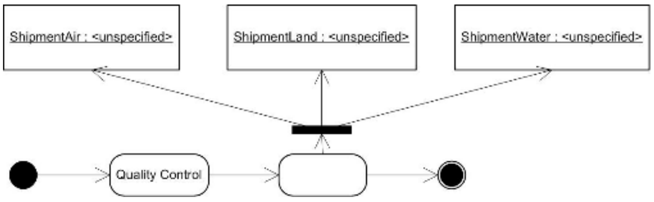

ShipmentAir : <unspecified>
ShipmentLand : <unspecified>
ShipmentWater : <unspecified>
Quality Control

Fig. 12.

Workflow Pattern: Multiple Instances Without Synchronization

Multiple Instance (MI) without synchronization defines the creation of multiple instances within an activity, all of them independent of each other. In addition, they may be able to execute in parallel.

```
Listing 8
<while condition="terminate">
        <invoke name="ShipmentAir"></invoke>
        <invoke name="ShipmentWater"></invoke>
        <invoke name="ShipmentLand"></invoke>
</while>
<receive name="ShipmentAir" createInstance="yes"></receive>
<receive name="ShipmentWater" createInstance="yes"></receive>
<receive name="ShipmentLand" createInstance="yes"></receive>
```

The code in Listing 8 invokes multiple instances of shipments in the while loop. To create an instance, each time the process is invoked, the `createInstance` attribute has to be set to “yes”. Notice, that this BPEL code is a theoretical approach and does not correspond to the visual representation in figure 1. It is an alternative solution, to better demonstrate this workflow pattern.

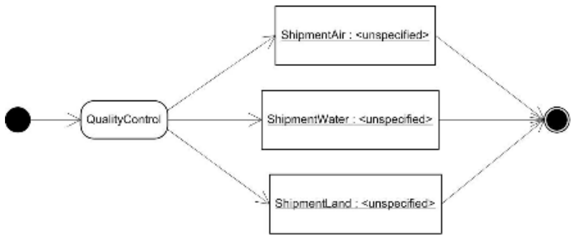

QualityControl
ShipmentAir : <unspecified>
ShipmentWater : <unspecified>
ShipmentLand : <unspecified>

Fig. 13.

Workflow Pattern: Multiple Instances with Synchronization

In contrast to the previous pattern, MI with synchronization means, that all instances, which are created in the scope of an action, are synchronized before they proceed with the workflow. This pattern is separated into several different structures, which differ from each other in the way the processes are instantiated. The first structure holds information about the number of instances at design time. In the second workflow pattern, the number of created processes is known at some stage of run time, but before the instantiation of the processes. The last pattern concerning MI with Synchronization does not know the number of instances to be created. New processes are created as they are needed, as long as no more instances are required. If the number of instances is known at compile time, the synchronization is comparable to the previous pattern, except for the need of an enclosing flow statement, which handles the synchronization.

```
Listing 9
<flow name="Shipping">
        <sequence><invoke name="ShipmentAir"/></sequence>
        <sequence><invoke name="ShipmentWater"/></sequence>
        <sequence><invoke name="ShipmentLand"/></sequence>
</flow>
```

The BPEL code sample in Listing 9 explains a possible solution for multiple instances, where the number of instances is known at run time. The three instances of Shipments are created and, enclosed within the flow statement, are synchronized.

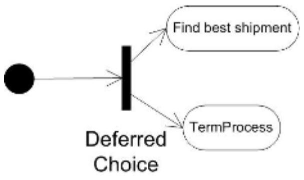


Fig. 14.

Workflow Pattern: Deferred Choice

In contrast to the Multi Choice pattern, this pattern chooses the branch to execute based on an event which is not necessarily available when the branch is reached. The decision, which branch to take is delayed until the suitable event has occurred. This construct is supported by the `<pick>` statement and is used especially for external triggers. We added a pick pattern to the logistics control application, as shown in Listing 10.

```
Listing 10
<sequence>
<pick>
<onMessage><invoke name="Find best shipment"/></onMessage>
<onAlarm><terminate name="TermProcess"/></onAlarm>
</pick>
</sequence>
```

This code extends the logistics control process shipment function. It is dynamically terminated by the terminate statement when a suitable solution was found. Sometimes it is not possible to find the best shipment in linear time, so we have to terminate the flow of control alternatively by a maximum amount of time, which is available for processing. During the consumption of the process the solution is approximated iteratively.

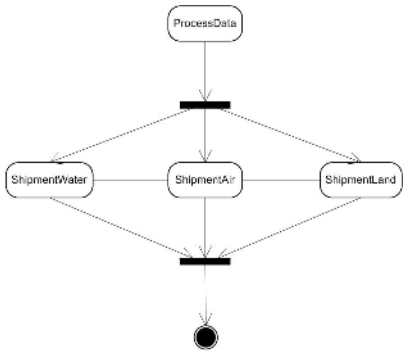


Fig. 15.

Workflow Pattern: Interleaved Parallel Routing

This workflow pattern defines a point in execution, where a set of activities is processed in a determined order. Each activity is executed only once and the order, in which they are processed, is defined at run-time. This pattern is realized in Collaxa with the `variableAccessSerializable` attribute in the `<scope>` statement. If this attribute is set to true, variable access in this scope is subject to concurrency control. In Listing 11 a BPEL code excerpt from the logistic process is shown.

```
Listing 11
<scope name="Shipment" variableAccessSerializable="yes">
<switch>
        <case><invoke name="ShipmentAir"/></case>
        <case><invoke name="ShipmentWater"/></case>
        <otherwise><invoke name="ShipmentLand"/></otherwise>
</switch>
</scope>
```

Listing 11 contains the process about the physical shipment for the parcels. They will be alternatively routed over sea, land or air. The visual representation in Figure 1 contains the flow statement enclosing the three different shipments to explain multiple instances with synchronization as explained in Figure 14.

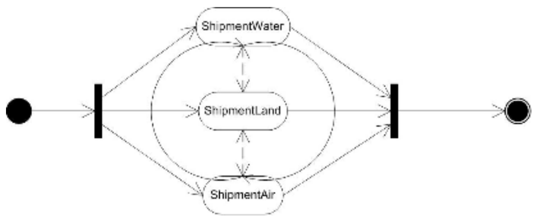


Fig. 16.

Workflow Pattern: Milestone

The Milestone workflow pattern defines a point in the workflow, where a determined milestone has to be reached to enable a given activity. A milestone can also expire which means that the activity will not be enabled. BPEL does not support this pattern directly. BPELJ [4] provides a possible solution for implementing a ‘workaround’. In Listing 12, a solution for this pattern is provided.

```
Listing 12
<bpelj:propertyAlias propertyName=″Token″ type=″bpelj:com.supply.Milestone″/>
   <bpelj:extractValue arg=″milestone″>
   milestone.getValue()
   </bpelj:extractValue>
</bpelj:propertyAlias>
<flow name="Shipping">
        <sequence><invoke name="ShipmentAir"/></sequence>
        <sequence><invoke name="ShipmentWater"/></sequence>
        <sequence><invoke name="ShipmentLand"/>
        <input part=″token″ variable=″milestone″/>
               <correlations>
         <correlation set=″Milestone″ initite=″yes″ pattern=″out″ parts=″token″>
        </correlations>
        </sequence>
</flow>
```

Listing 12 provides an example for a milestone pattern. In this code, ShipmentLand provides a token, which can be accessed externally by all other processes. If the milestone was achieved, execution will be advanced; otherwise the flow is stalled, until the Milestone will be reached by ShipmentLand. The only thing that is not standard BPEL in this example is the part attribute specified in the correlation element. In future BPEL notations, we expect the concept of *opaque correlations* whose values are chosen by the execution framework. This approach is not realized in Collaxa. Maybe it is subject for future work.

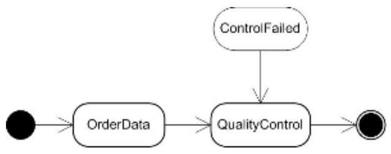


Fig. 17.

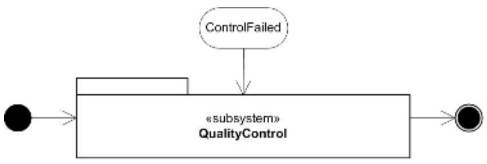


Fig. 18.

Workflow Pattern: Canceling

This pattern is divided into Cancel activity (Figure 17) and Cancel case (Figure 18). The first one terminates a running instance of an activity and the second one leads to the removal of an entire instance.

```
Listing 13
<scope name="QualityControl">
<sequence>
<flow>
        <sequence><receive name="ControlPassed"/></sequence>
        <sequence><terminate name="ControlFailed"/></sequence>
</flow>
</sequence>
</scope>
```

The code in Listing 13 aborts the execution of the process with the terminate action.

Table 1. Workflow pattern support in Collaxa

Workflow pattern	*Direct support*	*Workaround*	*not supported*
Sequence	X		
Parallel split	X		
Synchronization	X		
Exclusive choice	X		
Simple Merge	X		
Multi choice		X	
Synchronizing Merge		X	
Multi merge			X
Discriminator			X
Arbitrary cycles			X
Implicit termination	X		
MI without synchronization	X		
MI with synchronization	X		
Deferred choice	X		
Interleaved parallel routing	X		
Milestone		X	
Canceling	X		

4 Conclusion

In this paper we analyzed and systematically evaluated the support for workflow patterns in the Collaxa BPEL engine as defined in [5]. The majority of workflow patterns are supported by this system. Some of them are not even realizable in BPEL, but there are possibilities for workarounds which compensates this detriment. Some statements have restrictions which are known and expressed in the vendors Developer Resource. For example the `<invoke>` statement does not currently support local exception handling. To summarize the usability and design of the Collaxa server is straightforward. In contrast to other implementations the visual BPEL Designer makes the development of workflows easy compared to writing BPEL code directly. However, there does not exist a standard or "agreed upon" visual notation for BPEL. The preferred implementation architectures (JBoss[6] for Collaxa server and Eclipse[7] for BPEL Designer) rank as one of the most prominent of all related open source projects. Table 1 presents a summary of the discussed workflow patterns and their support in Collaxa. An 'X' in the cell indicates if (a) support of the current workflow pattern is given, (b) Collaxa provides possibilities for a workaround or (c) it is not possible to implement this pattern.

Acknowledgements. This research was supported in part by a research award for *Development Methods for Dynamic Web services Workflows* of the Chamber of Commerce Vienna (Wirtschaftskammer Wien).

References

1. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services. http://dev2dev.bea.com/techtrack/BPEL.jsp
2. http://www.collaxa.com
3. http://www.collaxa.com/pdf/cx-bpel-developer-20.pdf
4. Michael Blow, Yaron Goland, Matthias Kloppman et al. *BPELJ: BPEL for Java*, A Joint White Paper by BEA and IBM
5. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. *Workflow Patterns*, Distributed and Parallel Databases, 14, 5-51, 2003, Kluwer.
6. http://www.jboss.org
7. http://www.eclipse.org

Aspect-Oriented Programming for Dynamic Web Service Monitoring and Selection

Bart Verheecke, María Agustina Cibrán, and Viviane Jonckers

System and Software Engineering Lab
Vrije Universiteit Brussel
{Bart.Verheecke, Maria.Cibran}@vub.ac.be, vjoncke@info.vub.ac.be

Abstract. In Service-Oriented Application Development, applications are composed by selecting and integrating third-party web services. To avoid hardwiring concrete services in client applications we introduced in previous work the Web Services Management Layer (WSML) and suggested a redirection mechanism based on Aspect Oriented Programming (AOP). Even though this mechanism enables hot swapping between semantically equivalent services based on their availability, this is not enough to create applications that are driven by business requirements. In this paper we introduce a more advanced selection mechanism that allows dynamic switching between services based on business driven requirements that can change over time. Choosing a service may be done based on cost, presence on approved partners list, as well as binding support, quality of service classifications, historical performance and proximity. We introduce a modular monitoring mechanism that is able to observe these criteria and trigger a more advanced service selection procedure. We show how the AOP language JAsCo with its dynamically pluggable aspects is well suited to achieve this.

1 Introduction

Web service technologies accelerate application development by allowing the selection and integration of third-party web services, achieving high modularity, flexibility and configurability. However, as it is identified in [3], existing approaches used in state-of-the-art CASE tools, such as Microsoft Visual Studio.NET, typically hard wire service proxies in client applications. This makes it difficult to achieve just-in-time discovery and integration of web services. Furthermore, the issue of selecting the best or the most appropriate service from those that provide the required functionality is currently not addressed by these tools. As stated in [1], this leads to unmanageable applications that cannot adapt to changes in the business context. The Internet is an ever-changing and unpredictable environment; therefore, clients need to be flexible enough to deal with failures and changing network conditions, old services that are abandoned and new services that become available. The goal of our research is to fully decouple a client application from the concrete web services it uses. In our approach clients can be written in a service independent way, referring only to the functionality that is needed. As such, hot-swapping between services becomes possible: a service that is unavailable due to network conditions or service related problems can be easily replaced by a functionally equivalent one. While this mechanism allows high flexibility in the integration of web services, it is not enough to achieve applications that are driven by business requirements and need to stay competitive in a changing environment.

The aim of this paper is to extend this hot-swapping mechanism for the dynamic and optimal selection of services and service compositions taking into account business considerations. Our solution allows selecting web services based on the non-functional properties they advertise in their documentation or on their runtime behaviour which is monitored by dynamically introduced measurement points in the system. We show how *Aspect-Oriented Programming* (AOP) technology and in particular JAsCo [2] is well suited to realise the decoupling and modularisation of service selection policies and monitoring concerns.

The next section introduces the Web Services Management Layer as the context of this research. In section 17 we identify selection policies and motivate why AOP is well suited to modularize them. Section 4 introduces the main ideas behind AOP and JAsCo. Section 5 introduces our approach based on selection and monitoring aspects. A prototype implementation is discussed in section 6. Finally, section 7 presents related work and conclusions and future work are given in section 8.

2 Web Services Management Layer

To achieve a higher flexibility in Service Oriented Application Development (SOAD), we have presented in [3] [4] an intermediate management layer in between client applications and web services called **Web Services Management Layer (WSML).** In this earlier work we focused on eliminating the hard wiring of the services in client applications by introducing a flexible redirection mechanism which allows switching between multiple services and compositions that offer the same functionality. The WSML functions as a repository for services that can be used to functionally integrate with the application. Available services can be looked up in a central UDDI-registry or discovered using decentralised Web Services Inspection Language (WSIL) documents. The advantage of this approach is that the client application becomes more flexible as it can continuously adapt to the changing environment and communicate with new services that were not known or available at design time. Switching between services (or even service compositions) is done in a transparent way for the clients. This is done using Service Types, which are explained in more detailed in section 6.1. As such, the WSML actually realises dynamic integration of services.

In this paper, we suggest to extend our approach by adding support for dynamic and optimal selection of services based on business requirements.

3 Service Selection Criteria

In this section we identify selection policies that guide the just-in-time integration of web services in client applications and motivate the suitability of AOP to implement a flexible service selection and monitoring mechanism.

3.1 Identifying Selection Policies

In order to accomplish dynamic service selection based on changing business requirements, client applications need to define, as part of their requests, not only which functionality is needed but also which *non-functional requirements* are pursued. The consideration of non-functional requirements introduces the concept of the *most appropriate services*: the dynamic service selection mechanism should select the services that best fit the specified requirements and swap to them when needed. The non-functional requirements specified as part of the application requests imply the definition of *selection policies* used to govern the service selection. A *selection policy* defines a condition based on non-functional properties of services. These non-functional properties might need to be monitored during the execution of the application. Examples of non-functional properties of services are: response time, service cost, network bandwidth, service reliability and service dependability.

A service complies with a selection policy if it satisfies its condition. Only if a service complies with all specified selection policies it is *approved* to be considered during the selection process. An approved service can be selected and integrated in the application. If a service does not satisfy at least one selection policy, then it is *disapproved* and not considered for selection. Service selection policies can be classified as imperatives and guidelines. An **imperative** is a constraint on a service that should be satisfied at the moment the service is invoked. The constraint can contain absolute conditions (e.g. the cost of the service cannot exceed a fixed amount, the response time of a service cannot drop under some threshold), or can involve interrelationships with other services or the system (e.g. the cost of the service must be below the average of the cost of all registered services). If two or more services satisfy an imperative it is not determined which one must be chosen. This is where guidelines come in. A **guideline** specifies that if multiple services are available to provide the required functionality, one is preferred over the other (e.g. the cheapest service or the service with the highest encryption level must be selected). This implies that the services are compared with each other and a ranking is made.

3.2 Towards a Flexible Implementation of Selection Policies

After having identified what selection policies are, the challenge is how to implement them in a flexible way, avoiding the applications to change each time the business requirements change. We have conducted previous work [5] [6] [7] on the decoupling of business rules in the context of software applications developed using object-oriented or component-based software development techniques. These applications have substantial core application functionality and are normally driven by business policies. Thus, they need to constantly cope with changes in the business requirements embodied in their policies. In this context it is increasingly important to consider *business rules* as a means to capture some business policies explicitly. A business rule is defined by the Business Rules Group as *a statement that defines or constraints some aspect of the business. It is intended to assert business structure or to control the behaviour of the business* [8]. As business rules tend to evolve more frequently than the core application functionality [9] [10], it is crucial to *separate* them from the core application, in order to *trace* them to business policies and decisions, *externalize* them for a business audience, and *change* them. In that work, contrary to typical approaches in the field which only focus on decoupling the business rules themselves [11] [12], we observe the need to decouple the code that links the business rules to the core application functionality. This linking code *crosscuts* the core application and thus need to be separated in order to achieve highly reusable and configurable business rules. In this previous research we show how AOP (see section 4) is ideal to decouple the crosscutting business rules links. Moreover, the following requirements are pursued:

- connect business rules to core application events which depend on run-time properties,
- pass necessary business objects to events in order to make business rules executable at that event,
- reuse a business rule link at different events,
- combine, prioritize and exclude business rules in case of interference,
- control the instantiation, initialization and execution of business rule links.

Successful experiments using AspectJ [13] and JAsCo [2] (see section 4) were performed achieving the identified requirements. In the context of Service-Oriented Application Development, it should be possible to select and integrate the services that best accommodate to the application requirements. For instance, it may be required to integrate the cheapest service at a given time. Analogously to business rules, selection policies are driven by business requirements and need to dynamically reflect the changes in the environment. A possible situation is that the cheapest service is not needed anymore but the fastest as result of changing the business requirements. As a conclusion, selection policies should be kept separated from the services and applications that integrate them to enhance maintainability, reusability and adaptability. The same way AOP resulted ideal to separate business rules links and connect them with the core application, AOP can be successfully used to plug-in and out selection policies that govern the selection and monitoring of services.

4 Aspect-Oriented Programming

AOP argues that some concerns of a system, such as synchronisation and logging, cannot be cleanly modularized using current software engineering methodologies as they are scattered over different modules of the system. Due to this code duplication, it becomes very hard to add, edit and remove such a crosscutting aspect in the system. The goal of AOP is to achieve a better separation of concerns. To this end, AOP approaches introduce a new concept to modularize crosscutting concerns, called an *aspect*. An aspect defines a set of *join points* in the target application where the normal execution is altered. *Aspect weavers* are used to weave the aspect logic into the target application. Nowadays, several AOP approaches, such as AspectJ, Composition Filters [14], HyperJ [15] [16] and DemeterJ [17] are available. These technologies have already been applied on large industrial projects by for instance Boeing, IBM and Verizon Communications. For more information about AOP in general, we refer to [18].

However, a major drawback of most AOP approaches is that the aspects are woven with the core application at compile time. This would mean that every time the set of applicable business requirements changes, the application needs to be recompiled. Therefore, we need to use dynamic AOP technology. A suitable AOP language for achieving our objectives is JAsCo [2], a dynamic aspect oriented programming language that provides support to dynamically plug-in and out aspects in the application. Moreover, JAsCo aspects are described independently of a concrete deployment context, making them highly reusable in different concrete contexts. These characteristics make JAsCo suitable for web services applications and allow us to achieve the degree of flexibility pursued in the service selection. JAsCo is built on top of Java and basically introduces two new concepts:

- **Aspect Beans:** an aspect bean is an extension of the Java Bean component that specifies crosscutting behaviour in a reusable manner. It can define one or more logically related hooks as a special kind of inner classes. A hook specifies ***when*** the normal execution of a method should be intercepted and ***what*** extra behaviour should be executed at those points.
- **Connectors:** a JAsCo connector is responsible for deploying the crosscutting behaviour of the aspect beans and for declaring how several of these aspects collaborate. It specifies ***where*** the crosscutting behaviour should be deployed.

The dynamism in the creation of connectors constitutes JAsCo's key feature for the realisation of the dynamic service selection and integration as it is shown in the following section. For more information about JAsCo and the JAsCo component model, we refer to [2]. The use of JAsCo in the WSML is explained in section 6.

5 AOP for Dynamic Selection and Monitoring

We observe that selection policies and monitoring concerns are good candidates for aspects. After all, if web services are hard-wired in a client application, selection code needs to be included at each point where some service functionality is requested. This code results scattered in different places and tangled with code that solves other concerns of the application. Furthermore, the monitoring of services requires introducing measurement points at different points in the system. This monitoring code would not only be scattered and tangled, but also it would require the client application to anticipate the introduction of all these monitoring points. As such, all these points should be known at design time, which might not be the case.

To achieve the required flexibility, we propose to use **selection aspects** to nicely modularize selection policies. One selection aspect contains the code on how to select the most appropriate web service for a given business requirement. Possibly, these selection policies require the monitoring of unforeseen non-functional properties. Therefore, it should be possible to plug in and out additional monitoring functionality on demand. To accomplish this, we use **monitoring aspects** to observe system, environmental or service changes. For instance, if a selection policy specifies to use the fastest service available at any given time, the performance of the involved services needs be monitored by measurement points introduced by monitoring aspects. The monitoring points are precise moments in the execution of the system where the desired properties are affected. Monitoring aspects allow identifying these execution points and specifying which monitoring logic should be executed at exactly that time. This way, monitoring aspects allow the decoupling of monitoring from the applications. As such, AOP allows realizing a modularized non-invasive selection and monitoring mechanism. The use of selection and monitoring aspects in the context of the WSML is depicted in Fig. 1.

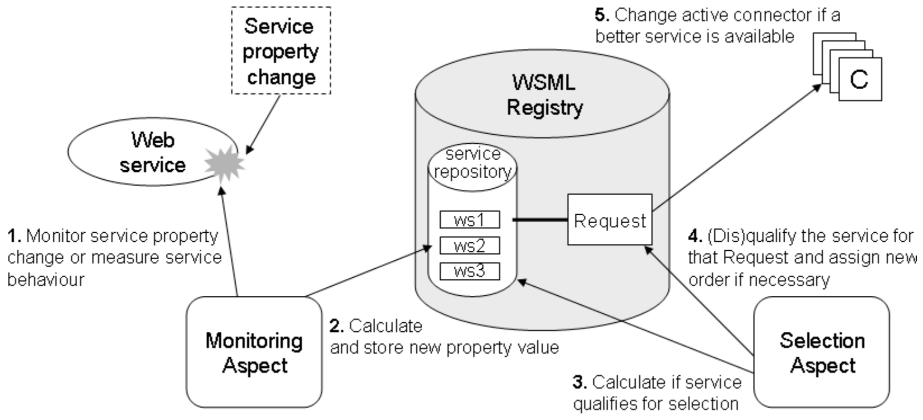

Service property change
Web service
WSML Registry
service repository
ws1
ws2
ws3
Request
C
5. Change active connector if a better service is available
1. Monitor service property change or measure service behaviour
4. (Dis)qualify the service for that Request and assign new order if necessary
Monitoring Aspect
2. Calculate and store new property value
3. Calculate if service qualifies for selection
Selection Aspect

Fig. 1. Service Monitoring and Service Selection Aspects in the WSML

6 Prototype Implementation

A fully implemented prototype of the WSML is available at [19]. It is developed in Java and uses JAsCo for implementing the identified aspects, since its features are well suited to achieve the required flexibility. This implementation is deployed and integrated with a platform of Alcatel Bell to realise a demonstrator in the domain of service provisioning in broadband networks.

6.1 Decoupling Web Services from Client Applications

Fig. 2 illustrates the overall architecture of the WSML using JAsCo aspect beans and connectors. To decouple the client application from specific web services, the notion of *Service Type* is introduced in the WSML: a generic specification of the required functionality without references to specific web services. A service type can be seen as a contract specified by the application towards the services and allows hiding the syntactical differences between semantically equivalent services. Two services may offer the same functionality but differ on a number of considerations like:

- Web method names
- Synchronous / Asynchronous methods
- Parameter types & return types
- Semantics of parameters & return values
- Method sequencing

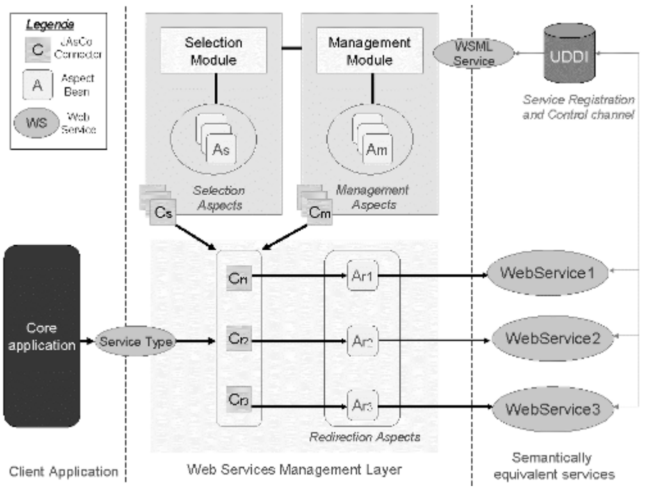


Fig. 2. Detailed Architecture of the WSML

Concrete services with different interfaces can be registered for a service type. Then, client applications can make a service type request and the WSML translates these generic requests to concrete web services invocations. In this mechanism we identify *redirection aspects*, which define the logic of intercepting client application requests and replacing them with concrete web service invocations. As such they encapsulate all *communication details* for a specific service or service composition. In addition, redirection aspects need to be dynamically plugged-in and out to reflect the volatility of the service environment. By creating a new connector and redirection aspect bean at runtime the new service or service composition can be integrated in the client application in a transparent way. The current version of the WSML supports full automatic generation of the connectors and provides tool support for the creation of the aspects. The proposed mechanism also enables hot swapping between services. If the response time of a service is too slow or the service becomes unavailable the selection module can "hot-swap" to another service by simply activating its connector and deactivating the previous one. Notice that this mechanism assumes simple request-response communications between the WSML and the Web Services: a hot-swap can take place at any time. If more advanced business processes need to be executed between services that keep state, swapping to another service becomes more difficult. It is clear that in this case compensation actions might be required to rollback the state of the service and maybe even the state of the application. A detailed discussion on this topic is however outside the scope of this paper.

6.2 Example Application

To illustrate these ideas, an example of a travel agency application is introduced. The application offers the functionality to book holidays online that customers can use to make reservations for both flights and hotels. To achieve this functionality, the application integrates several web services. Suppose *HotelServiceA* and *HotelServiceB* are services that offer the same functionality for the online booking of hotels. Each hotel service returns the same type of results. The client application has to present a list of available hotels to the customer. To this end a *HotelServiceType* is defined which specifies the following method: `List listHotels (Date, Date, CityCode)`. At deployment or run time the following two services are available:

HotelServiceA provides the method:
`HotelList giveHotels(CityCode,Date,Date)`
HotelServiceB provides the method:
`HotelList getHotels (Date,Date,CityName)`

6.3 Dynamic Service Selection and Monitoring

The goal of this section is to show how the WSML considers non-functional properties and Quality of Service constraints (QoS) to guide the selection of the most appropriate services. Selection criteria can be based on properties defined in the services descriptions and that can be retrieved and checked at the moment the criteria are applied (e.g. price, distance). Another possibility is that the properties involved in the selection criteria are not anticipated and defined in the services. These properties depend on the behaviour of the service at run-time. Examples of such properties are average response time, number of successful invocations, etc. Current approaches only provide limited or no support in the consideration of non-functional properties in the service selection. Following an example of an imperative selection policy that works on the properties of services is given. For instance, assume the hotel application wishes to communicate only with web services that can respond within 5000 milliseconds. As a consequence of this imperative, the distinction between approved and disapproved hotel services has to be made. Suppose the system did not foresee the functionality to actually monitor the response time of the web services. Using traditional software engineering approaches, it would not be straightforward to introduce and remove non-invasively the required measurement points at runtime. On the contrary, in our approach we can achieve this by defining a monitoring aspect that will calculate the average speed for each of the available services. The aspect needs to hook before and after the method execution of
`HotelServiceType.listHotels(args)` in the service type:

- before the method is executed, a timer is started
- after the method is executed, the timer is stopped, the average speed is calculated and stored.

This is depicted in figure 3 in lines 23 to 39. First, the aspect needs to be initialised in lines 3 to 10. There, the new property *AvSpeed* is added to each service and initialised with a start value *zero.* The same is needed for each new service that is registered later in the WSML. This is done in the *ServiceAddedHook* in lines 13 to 20.

```
1. class ServiceSpeedLogging {
2.    ...
3.    public void initialiseAspect (String ruleName,
4.       String serviceTypeName, String reqName) {
5.       ...
6.       for (int i=0, WSML.getNumberOfServices(), i++) {
7.          Service ws = WSML.getService(i);
8.          ws.addProperty ("AvSpeed", initValue);
9.       }
10.   }
11.
12    //Aspect Hook when a new service is added
13    hook ServiceAddedHook {
14.      ServiceAddedHook(method(Service ws)){
15.        execute(method);
16.      }
17.      after () {
18.        ws.addProperty ("AvSpeed", initValue);
19.      }
20.    }
21.
22.    //Aspect Hook to do the actual monitoring of the speed
23.    hook SpeedLoggingHook {
24.       private long start, stop = 0,
25.       private Service service;
26.
27.        //... hook constructor omitted here
28.        before () {
29.           service = WSML.getActiveService();
30.           start = System.currentTimeMillis();
31.       }
32.       after () {
33.          stop = System.currentTimeMillis();
34.          if (service == WSML.getActiveService()) {
35.              //calculate average speed
36.              service.setProperty ("AvSpeed", average);
37.           }
38.       }}}
```

Fig. 3. Monitoring aspect for determining the average speed of a service

After having defined the monitoring aspect which makes the necessary property available, a selection aspect can be defined to approve and disapprove the available hotel services based on their average speed. The WSML defines a library of reusable and generic selection policy templates which determines which web services to consider when satisfying a service type request. Fig. 4 shows a simplified version of the code for a basic selection policy template which approves and disapproves the services registered for a service type according to the values of a certain property.

```
class ServicePropertySelection {
  …
  public ServiceType serviceType;
  public List approved, disapproved;
  public Object maximum, minimum;
  public String property;

  // Aspect hook to approve or disapprove a service
  // when it is added for the first time depending on its
  // value of property
  hook WebServiceAddedHook {
    WebServiceAddedHook(method(WebService ws)){
      execute(method); }
    …
    after () {
       if (checkPropertyOfWebService(ws)) approve(ws);
           else disapprove(ws); }
  }

  // Aspect hook to approve or disapprove a service
  // whenever its value of property changes
  hook PropertyChangedHook {
    PropertyChangedHook(method(String mproperty,..args)){
      execute(method); }
    …
    after() {
      WebService ws = (WebService)calledobject;
      if (checkPropertyOfWebService(ws)) approve(ws);
      else disapprove(ws);
      // trigger the activation of the most appropriate
      // service for serviceType
      global.serviceType.activate(); }
    }

  // Aspect hook that computes the list of services
  // (among the ones registered for serviceType) that
  // satisfy the policy
  hook GetApprovedServicesHook {
    GetApprovedServicesHook (method(..args)){
      execute(method); }
    …
    replace() {
      List listPreFiltered = (List)proceed();
      List myApprovedServices = global.getApproved();
      myApprovedServices.retainAll(listPreFiltered);
      return myApprovedServices; }
   }}
```

Fig. 4. Selection policy aspect for the approval of services according to values of a property

In the `ServicePropertySelection` aspect, the hooks `WebServiceAddedHook` (lines 11 to 18) and `PropertyChangedHook` (lines 22 to 33) approve or disapprove a web service depending on the value of the property the policy is defined upon. This behaviour is executed when the service is added for the first time or when the service changes the values of its pertinent property respectively. The hook `ActivationListHook` (lines 38 to 47) retrieves from the list of services registered for `serviceType`, only the services that satisfy the policy. Note that in the case more than one policy is defined, the different selection aspects need to cooperate to come up with only one list of services that satisfy all the policies. If this is the case, the `proceed()` (line 43) proceeds with the enforcement of the other policies and afterwards, on this pre-filtered list of services, only the ones approved by the current policy will be kept and considered for activation by the WSML.

Because of the reusability of JAsCo aspect beans, a selection policy aspect can be instantiated and initialised with different parameters at runtime by creating new connectors. In order to define a selection policy which filters services according to their average speed (property captured in the monitoring aspect presented in figure 3), the ServicePropertySelection aspect needs to be deployed with the specific values: `serviceType` = HotelServiceType, `minimum` = 0, `maximum` = 5000, `property` = "avSpeed", as well as the abstract methods of the hooks need to be linked to the concrete methods for the addition of a web service, the setting of a web service property and the listing of registered web services respectively. This deployment is done in a JAsCo connector (omitted here for space reasons). Note that if a new service is added in the WSML, the *average speed* property will not be set until it is actually invoked. To avoid new services to never have the chance to be invoked, as they have no reputation yet, their selection could be based on the recommendations or endorsement by trusted third parties.

Other reusable selection policy templates are provided in the WSML, such as a policy which orders the approved services according to the values of their properties and a policy ensuring the addressing of always the same service for the requests in a transaction. These policies can be deployed dynamically in JAsCo connectors, allowing a wide variety of selection policies to be specified and reinforced at runtime. Note that, even if only shown in this paper for service types, selection policies can also be specified individually at the level of each service type request to realise a more fine-grained selection.

For each service policy, one connector and one aspect bean instance is created. This could imply a certain overhead if many selection policies need to be enforced at run-time for a given service type. In JAsCo, an aspect-oriented just-in-time compiler called Jutta [20] is provided to optimize the aspect interpreter system and hence enhance performance. It is important to consider that sometimes searching for the "best" solution could be computationally expensive. Moreover, because the "best" may change frequently over time, unwanted oscillations can seriously affect performance. In this regard, selection policies could be useful as a mechanism to prevent undesired oscillations between services.

7 Related Work

A lot of research is going on in the web service context and a lot of vendors are currently working on dedicated web service management platforms. However, most of these products focus on the server-side management of web services. They allow developers to build and deploy web services and also provide some management capabilities such as load balancing, concurrency, monitoring, error handling, etc. Our approach provides support for the client applications that want to integrate different third-party web services and manage them. Some of the most relevant approaches are: The Web Service Description Framework (WSDF) [21] incorporates ideas from the Semantic Web [28] suggesting an ontological approach for the side effect free invocations of services. The Web Services Mediator (WSM) [22] also identifies the need for a mediation layer to achieve dynamic integration of services. However, as far as we know there is no implementation available. The Web Services Invocation Framework (WSIF) supports a Java API for invoking web services irrespective of how and where the services are provided. WSIF mostly focuses on making the client unaware of service migrations and change of protocols.

A related approach to enable more advanced web service selection is agent-based architectures. Agents are autonomous, computational entities that perceive their environment through sensors and act upon their environment through effectors [23]. As such, they can be used to monitor and select services in a non-invasive manner. In [24] a solution is suggested to collect, aggregate and disseminate rating information on the usage of web services by proxy agents. It uses a social algorithm to make selections. A prototype of this system is under development.

The idea of applying AOP concepts to decouple web services concerns is quite innovative and thus not many approaches have been developed focusing on this field. However, Arsanjani et al. [25] have recently identified the suitability of AOSD to modularize the heterogeneous concerns involved in web services. However, they refer to approaches like AspectJ [13] and the Hyper/J [15] [16] which only allow static aspect weaving though, contrary to JAsCo which supports dynamic pluggability of aspects. They also identify the need for the web services to include the definition of non-functional interfaces that permit control over performance, reliability, availability, metering and level of service.

8 Conclusions and Future Work

In this paper we show how dynamic service selection can be realised as part of the WSML using AOP. We identified the need to consider service criteria to drive dynamic service selection and showed how JAsCo, a dynamic AOP language, can be successfully used for its realisation. JAsCo aspects are used to enhance the hot-swapping mechanism with the consideration of service criteria that can be added dynamically and to monitor service properties that will affect the selection. The advantage of using JAsCo is that the creation, addition and removal of aspects can be done at run-time, achieving the desired flexibility needed in the evolving and changing web services environment.

In state-of-the-art distributed computing infrastructure supporting Web Services, the service functionality is written down in terms of XML syntax without a well-defined semantics, such as WSDL, XLANG, WSFL or the more recent WS-BPEL [26]. The Semantic Web project [27] addresses these issues by developing a rich language allowing to describe all the complex interrelationships and constraints between services and other web objects. During the discussion in this paper we specify selection criteria using universal terms assuming that both the WSML and the Web Services talk the same language and understand each others capabilities. However, to realise this outside an experimental scenario and deploy the WSML in a complex heterogeneous environment like the internet, such a language is highly needed. Several ontology languages are currently under development, they include the Resource Description Framework [28], the Web Ontology Language [29] and DAML+OIL [30]. While this paper focuses only on the technical part of using AOP for advanced selection, a semantic web language can be adopted by the WSML to unambiguously describe all service criteria. Note that this also requires that the web services themselves must be described in this expressive language. OWL-S (formerly known as DAML-S) [31] is an example of such a language, using the DAML+OIL ontology.

This implementation of the WSML can be continued in different directions. In the implementation we have presented, selection aspects and their connectors are written by hand. There are two lines of work in this respect: either to choose one of the formal languages based on ontologies to specify the selection criteria and to (semi) automatically translate these descriptions into aspects and connectors. To achieve this, automatically code generation for the aspects and connectors is required. The other alternative is to provide a library of well defined aspect templates that can be easily deployed and used by the developer of the client application through wizard-based tool support.

So far, the selection aspects suggested are based on service properties that can be queried or monitored. Other kinds of service criteria involving other services characteristics (not necessarily properties) can be considered. For instance, we can imagine that we want to select services depending on certain behavioural patterns or dependencies with other services. Also the service selection can be extended to perform the service choice depending on conditions applied to the result of service invocations. The identification and definition of other selection aspects to contemplate these cases is subject of ongoing research.

Acknowledgements. The research presented in this paper is conducted with the support of the Flemish Government Project MOSAIC, funded by the "*Institute for the Innovation of Science and Technology in Flanders*" (IWT) or in Dutch "*Instituut voor de aanmoediging van Innovatie door Wetenschap en Technologie in Vlaanderen*", and is carried out in cooperation with Alcatel Bell.

References

[1] J. Malhotra, Ph.D., Co-Founder & CEO interKeel Inc., “Challenges in Developing Web Services-based e-Business Applications,” Whitepaper, interKeel Inc., 2001

[2] D, Suvée, W. Vanderperren and V. Jonckers, “JAsCo: an Aspect-Oriented approach tailored for Component Based Software Development”, Proc. of Int. Conf. on Aspect-Oriented Software Development (AOSD), Boston, USA, pp 21-29, ISBN 1-58113-660-9, ACM Press, March 2003.

[3] B. Verheecke, M. A. Cibrán and V. Jonckers, “AOP for Dynamic Configuration and Management of Web Services,” Proc. of ICWS'03-Europe, Erfurt (Germany), September 2003. To be published in the Int. Journal on Web Services Research (JWSR), Vol.1 No. 3, July-September 2004.

[4] M. A. Cibrán, B. Verheecke and V. Jonckers, "Modularizing Client-Side Web Service Management Aspects", Proc. of 2nd Nordic Conf. on Web Services (NCWS'03), Växjö (Sweden), November 2003

[5] M. A. Cibrán, M. D’Hondt and V. Jonckers, “Aspect-Oriented Programming for Connecting Business Rules”, 6th Proc. of Int. Conf. on BIS, Colorado Springs, USA, 2003.

[6] M. A. Cibrán, M. D'Hondt, D. Suvée, W. Vanderperren and V. Jonckers, “JAsCo for Linking Business Rules to Object-Oriented Software”, Proc. of Int. Conf. on CSITeA, Rio de Janeiro, Brazil, June 2003.

[7] M. A. Cibrán, D. Suvée, M. D’Hondt, W. Vanderperren and V. Jonckers, “Integrating Rules with Object-Oriented Software Applications using Aspect-Oriented Programming”, to be published in Proc. of the 5th Argentine Symposium on Software Engineering (ASSE’04), Córdoba, Argentina, September 2004.

[8] The Business Rules Group. “Defining Business Rules: What Are They Really?”, http://www.businessrulesgroup.org/, July 2000.

[9] G. Kappel, S. Rausch-Schott, W. Retschitzegger, and M. Sakkinen, “From rules to rule patterns”, Proc. of Int. Conf. on Advanced Information Systems Engineering, pp. 99-115, 1996.

[10] B. von Halle, “Business Rules Applied”, Wiley, 2001.

[11] B. N. Grosof, Y. Kabbaj, T. C. Poon, M. Ghande, and et al., “Semantic Web Enabling Technology (SWEET)”, http://ebusiness.mit.edu/bgrosof/

[12] I. Rouvellou, L. Degenaro, K. Rasmus, D. Ehnebuske and B. McKee, “Extending business objects with business rules”, TOOLS Europe 2000, St-Malo, France, pp. 238-249, 2000.

[13] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersen, J. Palm, and W. G. Griswold, “An overview of AspectJ”, Proc. of ECOOP’2001, volume 2072 of Lecture Notes in Computer Science, pp. 327-353, Berlin, Heidelberg, and New York. Springer-Verlag.

[14] L. Bergmans, M. Aksit, “Composing Crosscutting Concerns Using Composition Filters”, Communications of the ACM, Vol. 44, No. 10, pp. 51-57, October 2001.

[15] H. Ossher, P. Tarr, “Using multidimensional separation of concerns to (re)shape evolving software”, Communications of the ACM, 44(10):43-50, October 2001.

[16] P. Tarr, H. Ossher, W. Harrison, S. Sutton, “N degrees of separation: Multi-dimensional separation of concerns”, ICSE 1999, pp.107-119. IEEE Computer Society Press/ACM Press.

[17] K. Lieberherr, D. Orleans, J. Ovlinger, “Aspect-oriented programming with adaptive methods”, Communications of the ACM, Vol. 44, No. 10, pp. 39-41, October 2001.

[18] Communications of the ACM, “Aspect-Oriented Software Development”, October 2001.

[19] Web Services Management Layer (WSML), http://ssel.vub.ac.be/wsml/

[20] Vanderperren, W., Suvee, D. “Optimizing JAsCo dynamic AOP through HotSwap and Jutta”, Proceedings of Dynamic Aspects Workshop published as RIACS Technical Report 04.01., Lancaster, UK, March 2004.

[21] A. Eberhart, “Towards Universal Web Service Clients”, Proc. Euroweb 2002, UK.
[22] S. Chatterjee , “Developing Real World Web Services-based Applications”, The Java Boutique, http://javaboutique.internet.com/articles/WSApplications/
[23] G. Weiss, “Multi agent Systems, a Modern Approach to Distributed Artificial Intelligence”, The MIT Press, Cambridge, Massachusetts, 1999, isbn 0-262-23203-0
[24] E. M. Maximilien, M. P. Singh, “Agent-based Architecture for Autonomic Web Service Selection”, Proc. of Workshop on Web Services and Agent-based Engineering, Melbourne, Australia, 2003
[25] A. Arsanjani, B. Hailpern, J. Martin, P. Tarr, “Web Services Promises and Compromises”, ACM Queue, Vol. 1 No. 1, March 2003 http://www.acmqueue.org/
[26] BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, “Business Process Execution Language for Web Services (BPEL4WS) v 1.1”, May 2003
[27] T. Berners-Lee, J. Hendler, O. Lassila, “The Semantic Web,” Scientific American, vol. 284, no. 5, May 2001, pp. 34-43
[28] Resource Description Language http://www.w3.org/RDF/
[29] F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein “OWL Web Ontology Language Reference”, W3C Working Draft 31 March 2003.
[30] D.L. McGuinness, R. Fikes, J. Hendler, L.A. Stein., “DAML+OIL: An Ontology Language for the Semantic Web,” IEEE Intelligent Systems, vol. 17, no. 5, Sept./Oct. 2002, pp 72-80.
[31] A. Ankolekar, M. Burstein, J. R. Hobbs,O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K.Sycara, “DAML-S: Web Service Description for the Semantic Web,” The Semantic Web – ICWC 2002: Proc. 1st Int. Semantic Web Conference (ISWC), Springer-Verlag, Berlin 2002, pp. 384-363

Semantically Extensible Schemas for Web Service Evolution

Erik Wilde

Swiss Federal Institute of Technology Zürich (ETHZ)
Paper URL: `http://dret.net/netdret/publications\#wil04j`

Abstract. Web Services are designed for loosely coupled systems, which means that in many cases it is not possible to synchronously upgrade all peers of a Web Service scenario. Instead, Web Service peers should be able to coexist in different versions. Additionally, older software versions often could benefit from upgrades to the service if they were able to understand it. This paper presents a framework for semantically extensible schemas for Web Service evolution. The core idea of is to use declarative semantics to describe extensions to a service's vocabulary. These declarative semantics can be used by older software versions to understand the semantics of extensions, thus enabling older software to dynamically adapt to newer versions of the service. As long as declarative semantics are sufficient, older software can benefit from the service's extension.

1 Introduction

In this paper, a framework for semantically extensible schemas for Web Service evolution is presented. It is based on various technologies related to the *Extensible Markup Language (XML)* [1]. The basic idea is to construct a framework which augments the almost non-existent support for versioning of Web Services in the *Simple Object Access Protocol (SOAP)* and the *Web Service Definition Language (WSDL)* [2]. Versioning not only covers controlled ways to deal with different version of a service's vocabulary, but also means to semantically describe extensions, so that older software versions can "understand" newer versions of the service vocabulary.

In general, Web Services claim to be middleware for distributed applications, and they also claim to be particularly well-suited for Web-style scenarios, with the most important characteristic being *loose coupling*. Loose coupling means that the components of such a system should be as independent as possible, and this includes the independent evolution of individual components. In this paper, we present an approach which makes it easier for Web Service designers and users to achieve loose coupling. Whereas we describe this approach as an separate concept, based on certain schema design guidelines, we hope that future versions of Web Service specifications will provide better built-in support for this type of service evolution.

The Web's ongoing development is an excellent example for loose coupling, because for the successful evolution of the Web it is crucial that the components (content providers on the server side and Web browsers as clients) can evolve individually. It is unrealistic to expect that servers and clients are always updated simultaneously, and one of the most important success factors of the Web is its ability to deal with version mismatches in a graceful way. This is described in more detail in Section 2, describing compatibility issues, and Section 4.1, giving an HTML-oriented example of how to implement different strategies for dealing with version mismatches.

Even though Web Services in their most general sense can be regarded as services as ubiquitous and easily accessible as Web servers, in reality most Web Service scenarios are closed application areas, regulated through companies or consortia which make change management much easier than in an open environment. However, as Web Services grow more popular, are used by larger communities, and exist for longer times, service evolution and associated change management will become an important issue.

The general question of how XML vocabularies can be versioned has been discussed in a W3C document by Orchard and Walsh [3] and is still an active field of research. In this paper, the question of vocabulary and schema design is discussed in Section 3, which looks at XML vocabularies from the perspective of XML Schema only. More generally, approaches to improve semantic interoperability through various mechanisms have been published by Su et al. [4], Hunter [5], Castano [6], and others, because semantic interoperability is a interdisciplinary field with many possible applications areas. However, the specific issue of how to leverage semantic interoperability for Web Service evolution so far has not been investigated in detail.

2 Compatibilities

The focus of this paper is how to incorporate features for Web Service evolution into schema design. In order to discuss this issue more specifically, Web Service evolution as the parallel development, deployment, and operation of Web Service servers and clients must be regarded for the two possible types of versioning issues:

– *Backward Compatibility:* This is the easier variant of compatibility, which requires that a newer version of a program can interpret and use the data formats of older versions. Backward compatibility can be implemented fairly easily by requiring that every new revision of a program must be able to deal with the data format implemented by older revisions.
– *Forward Compatibility:* This type of compatibility is harder to achieve, and in many real-life examples is not implemented in a usable way. Forward compatibility means that an older version of a program should be able to interpret and use newer data formats, either by ignoring unknown extensions (*weak forward compatibility*), or by being able to interpret extensions (*strong forward compatibility*).

> The challenge for forward compatibility is to deal with possible future versions without knowing ahead what they will look like. Naturally, this is impossible in an unrestricted scenario, where future versions are unrestricted in their ability to change the data format. However, if the data format is based on a schema that provides reasonable extension points and strategies, then it is possible to achieve forward compatibility.

Translated into the world of Web Services, backward and forward compatibility often appear in parallel, because most Web Services are request/response-based. So if a server is upgraded, then the server must be backward compatible to understand requests from older clients, and clients must be forward compatible to understand the server's responses.

Since backward compatibility is easy to achieve, it is not considered in this paper. Forward compatibility, on the other hand, is a challenging issue and the focus of this paper. In the vast majority of Web Service scenarios, there are many more clients than there are servers. Since the absence of forward compatibility means that a server can only be updated after all clients have been updated, too, it is desirable to avoid this kind of dependency. Compatibility in both directions guarantees a maximum degree of independence of servers and clients, and since the concept of Web Services is based on the idea of loosely coupled systems, this independence is very important.

Forward compatibility means that older clients must be able to interpret data from newer servers, without the requirement of an complex version negotiation and data transformation process. Also, it may be required for clients to actually use new data that has not been part of the schema when the client was implemented. Thus, dealing with extensions can be done in two ways:

– *Ignoring Unknown Data (weak forward compatibility):* The simplest strategy to deal with unknown extensions to the known data format is to ignore them. This (1) is only possible if they can be identified (which can be easily achieved with XML), and it (2) should only be done if ignoring extensions does not affect the semantics of the interpreted data.
  Well-known technologies using this kind of strategy are the rules for interpreting HTML's `object` element (rendering the object *or* the element's content), and SOAP's `mustUnderstand` attribute (flagging mandatory header blocks).
– *Interpreting Self-describing Data (strong forward compatibility):* Since ignoring unknown data often is not good enough (and only a very primitive way of dealing with extensions), a more interesting and promising way of reasonably implementing forward compatibility is to interpret extension data that is self-describing[1]. This strategy requires a framework for semantically describing data, and this question is discussed in Section 4.1.

[1] XML itself is often said to be "self-describing", but this is only true syntactically, since XML does not carry any semantics. If XML data should be able to convey self-describing semantics, an additional framework for this kind of information is required.

Basically, forward compatibility is about “graceful degradation”. Looking at the two extremes, one strategy is to ignore all data belonging to unknown extensions, a behavior which in many cases will not be accepted as being “graceful”. The other extreme is that the self-describing data can be interpreted fully, without any compromises regarding the extension semantics. In this case, the new extension can be perfectly handled by the older application, and in essence, no application update is be required, at least not from the extensibility point of view. In reality, however, many extensions fall between these two extremes, they are not fully ignored, but they are also not fully supported by earlier application versions. This difference between the intended behavior when dealing with an extension and the “graceful degradation” of older application versions is discussed in Section 4.3.

3 Schema Design Issues

Even though WSDL in principle can be based on any schema language, in practice in uses *XML Schema* [7] as its schema language. In its `types` section, WSDL defines the types that are used for exchanging XML messages. WSDL does not have an inherent way of versioning (nor do SOAP or UDDI), and the question so far has not been subject of any research activities. Thus, versioning WSDL descriptions is entirely within a user’s responsibility, as is the corresponding management of schema information.

The core question of versioning Web Services is that of versioning XML Schemas. The two important issues are to design *open* and *extensible* schemas, which then can be used as a foundation for a Web Service extension framework. These two issues are defined as follows:

– *Openness:* An open schema allows unknown data to appear at certain points within an instance, so that instances of the schema may use extensions’ instance data, but are still valid instances of the original schema.
  Since the goal of Web Service evolution is full compatibility, including forward compatibility, it is necessary to design the schemas in a way so that instances of extensions are also valid instances of earlier versions. To make this possible, a schema must be *open*, which means that it allows content in certain places without knowing the exact schema of the content. In basic XML, this can only be achieved through the DTD’s `ANY` content specifier[2]. XML Schema provides *Wildcards* for making a schema open. Using XML Schema’s `any` and `anyAttribute` elements, it is possible to open a schema for unknown elements and attributes. Wildcards can be restricted to allow elements or attributes from certain namespaces only, which is discussed at the end of this section. The `processContents` attribute of wildcards can be used to specify whether wildcards should be validated; for the Web Service evolution scenario it should be set to `lax`, requiring validation if and only if a schema is available.

[2] However, `ANY` does not allow real openness, because it only allows any *declared* element.

– *Extensibility:* An extensible schema is designed with versioning in mind, so that future revisions of the schema have clear ways of extending the schema. Versioning is built into the schema, and the schema's documentation makes clear how versioning should be implemented.

  Basically, there are two different issues when looking at extensibility. The semantics of an extension are unknown in advance, and have to be described using the declarative semantics described in Section 4. The schema of the extension, however, has to be described in XML Schema, and through its type layer, extensibility can be achieved in different ways.

  Using type substitution, an instance may substitute the type of an element with a type derived from the original element's type. This substituted type must be declared in the instance with an `xsi:type` attribute. This mechanism can be used to use a schema in a more variable way than explicitly allowed by the schema. A limitation of this mechanism is that only elements may be substituted.

  Using the `substitutionGroup` attribute of an element declaration, an element may declare itself as a member of the substitution group of another element. In this case, the substitution group member may be used in all places where the substitution group head is allowed[3]. Basically, substitution groups achieve an effect very similar to type substitution, but they are explicitly declared in the schema. Since substitution groups operate on element declarations, only elements may be substituted.

Openness and extensibility are issues that must be kept in mind when designing the first version of a schema, otherwise it will become unnecessarily hard to implement versioning. Through the `block` and `final` attributes of element and type declarations, it is possible to control type substitution and substitution groups. It is important to notice that XML Schema's default is to allow everything, which should be avoided by using the `blockDefault` and `finalDefault` attributes for the `schema` element and thus disallow everything, except for cases where these mechanisms are explicitly required and appropriate support is implemented.

As a side note, it has to be mentioned that much of the reuse of XML Schema is based on types, and the consequence of this is that processing of XML Schema instances should be type-based rather than name-based[4].

One last important consideration for XML Schema design is the use of *XML Namespaces* [8]. Namespaces can be structured in various ways, in some

[3] When using substitution groups, XML Schema's *identity constraints* become problematic, because they are based on element names, while the appropriate handling of substitution groups requires access to the type information. In some cases, this can be circumvented by using cleverly designed XPaths for the identity constraints, but in general substitution groups and identity constraints do not mix very well and should not be used in the same scope of a schema.

[4] Unfortunately, accessing type information is not properly supported through many XML APIs.

cases only containing XML element and attribute names, in other cases also containing additional subsets of names (such as for XML Schema, where a schema's namespace also contains type and other names). Namespace names must be syntactically correct URIs, but there is no need to actually associate a resource with a namespace's URI. However, it is useful to associate some descriptive resource with a namespace's URI, and a reasonable candidate for this is the *Resource Directory Description Language (RDDL)* or another language combining human-readable with machine-readable information.

The most interesting question when using XML Namespaces in a versioning scenario is whether different versions should use the same or different namespace names. There is no standard mechanism for namespace versioning, so the standard itself provides no support for declaring that an extension's namespace extends another namespace. However, when using a namespace description facility such as RDDL or something similar, it is reasonable and easily possible to include namespace dependency information in a namespace's description. In this case, using different namespace names for schema extensions are the preferable alternative. Whatever the namespace management approach may be (single namespace or namespace versioning), it is crucial to integrate this approach with the schema's wildcard declarations.

4 Extension Semantics

When discussing the semantics of schema extensions, two different forms of semantics must be clearly separated [9]. The first form of semantics are the *declarative semantics* that are hardwired into the extension framework through a declarative language, and are believed to be sufficiently expressive to cover the major issues of extensions. The second form are the *non-declarative semantics* (often also referred to as *procedural semantics*), which cover all aspects of extensions' semantics that cannot be captured by the declarative semantics.

Figure 1 shows the different forms of extension semantics. It also shows an application's *initial semantics* (I), which are the semantics of the application's initial vocabulary. Without semantic extensibility, an application is always limited to these semantics when dealing with unknown extensions. Through the usage of *declarative semantics* (II), however, it is possible to declaratively extend the semantics supported by an application to previously unknown extensions (IIa, IIb, and IIc). These declarative semantics are discussed in Section 4.1.

The second form of semantics are *non-declarative semantics* (III), which are the complement of the declarative semantics and thus together with the latter cover the "true" or "full" semantics of extensions. These semantics are discussed in Section 4.2. In order to find the right balance between expressiveness and complexity of declarative semantics, it is important to understand the difference between the two forms of extension semantics, and to deal with this difference in a reasonable way, so that there is the optimal balance between the extensibility of the schema, and the declarative expressiveness of the extensions' semantics. This question is discussed in Section 4.3.

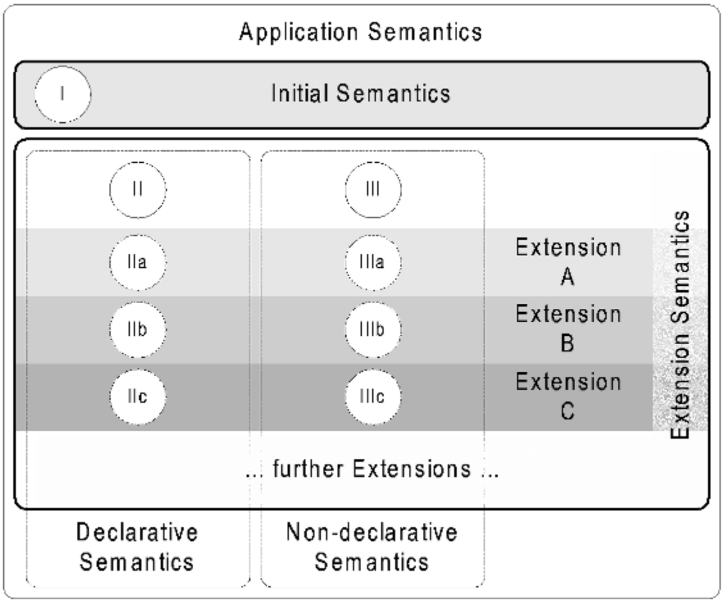


Fig. 1. Extension Semantics

4.1 Declarative Semantics

Since the goal is to make schemas semantically extensible (in addition to the simple vocabulary extensions discussed in Section 3), there must be a framework for describing semantics. This framework can range from extremely simple labels (e.g., `mustUnderstand` and/or `mayIgnore` labels) to very complex semantics, defining aspects of the extensions that may be used for different processing steps. This paper does not discuss the details of declarative semantics design, but instead focuses on the management and distribution of this information.

However, it should be kept in mind that semantics describe the meaning of extensions, and the meaning depends on the intended application (there is no meaning without context). This means that there can be different semantics for different facets or processing steps of the application scenario, which then are reflected in multiple semantics declarations for extensions.

Figure 2 shows an example of multiple declarative semantics (II′, II″, and II‴). It is also shown that not every extension must use all declarative semantics. If in the application context it is sufficient for an extension to be described using a subset of the available declarative semantics, then this is perfectly acceptable. Any extension using only a subset of the available declarative semantics then can only be interpreted in the context of these particular semantics.

Declarative semantics are necessarily represented in some form of formalism, and the ability to interpret this formalism is required for all components of

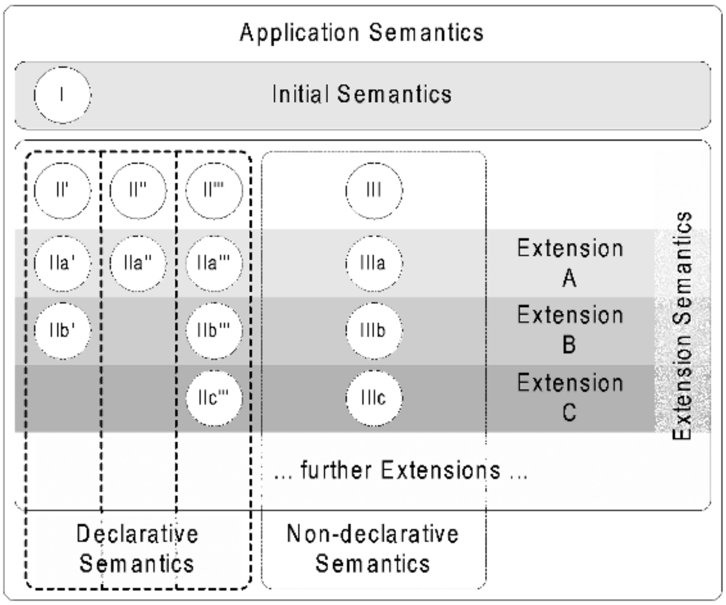


Fig. 2. Multiple Declarative Semantics

the extension framework. Without this ability, extensions can not be described semantically[5]. In case of the extension framework presented here, there is the question of whether to encode the semantic information within instance data (in which case we speak of *extensional semantics*), or within the schema (called *intensional semantics*):

– *Extensional Semantics:* In this case, the semantics declarations are part of the instance data[6], so that each instance is completely self-describing, at least with respect to the declarative semantics. The advantage of extensional semantics is the ability to access the semantics by interpreting the instance, rather than having to access the schema. However, there is a large overhead associated with this, in particular if the semantics declarations are non-trivial. Another drawback is that the application has to trust the instance originator (rather than the schema originator) to describe the semantics, which can be a severe security issue.
– *Intensional Semantics:* Since extensions are described in the schema (rather than individually by instance authors), it makes more sense to describe their semantics in the schema, too. In this case, extensions in an instance cannot

[5] Of course, if an implementation does not need to understand all kinds of declarative semantics, if there is more than one, it is possible to have implementations supporting only a subset of the existing declarative semantics.
[6] Naturally, the schema must define some mechanism to represent this information.

be interpreted semantically without accessing the extension schema, which is the drawback of the intensional approach.

There are drawbacks and advantages for both type of declarative semantics, but for most scenarios, the intensional approach is preferable, because it avoids the duplication of semantics declarations in instances, and because it avoids some of the security issues of the extensional approach. Furthermore, the extensional approach is best suited for cases where instance originators need to make up ad-hoc extensions for individual instances, which in most application scenarios is not a requirement.

However, there is one important exception to the general observation that intensional semantics are preferable over extensional semantics, and this is the case when semantics absolutely need to be observed, even in cases where the application can or does not want to access the intensional semantics. The most popular example for this kind of semantics is the `mustUnderstand` semantics, which signals to the application that it must be able to process the semantics of this extension[7]. Since this kind of information is crucial to the flawless function of the extension framework, it must be represented by extensional semantics.

In order to demonstrate these types of extensions, it is interesting to look at HTML, which supports an interesting combination of extension mechanisms. HTML provides hooks for embedding semantics (in case of HTML, these are formatting semantics) into an instance, but also makes it possible to describe the semantics of extensions in separate resources, which are referenced by instances. Since it is not quite clear whether these separate resources should be regarded as an external part of the instance or of the extension schema (if there is one)[8], this case is labeled as "borderline" in Figure 3, which shows the possible combinations of HTML's mechanisms.

The example for extensional vs. intensional extensions shown in Figure 3 assumes (for demonstration purposes only) that a new element `textbox` should be introduced in HTML for separate text boxes (in fact, flowing "textual figures" just like Figure 3 itself). For the first extensional case, the extension is implemented as an existing HTML element (HTML's `div` element, which does not have any inherent formatting semantics) and directly associated style information. The style information uses CSS, which is the declarative formatting semantics language for HTML. There is no guarantee that every intended extension to HTML can be described through CSS declarations, but the CSS vocabulary covers a broad area of formatting semantics, so that many extensions can in fact be described using CSS. Version 2 of the extensional case shows a variation, using the `class` mechanism supported by HTML and CSS. From the semantics

[7] Whether the declarative semantics are sufficient or not depends on the application. Consequently, in many cases it probably makes sense to differentiate between `mustUnderstandDeclaratively` and `mustUnderstandFully`.

[8] The most reasonable way to decide this is probably the author of the external resource. If the external resource has been provided by the schema author, then the instance uses intensional semantics, and extensional semantics otherwise.

"Extensional" HTML Extension (Version 1):

```
<div style=" [[ CSS Declarations ]] "> ... </div>
```

"Extensional" HTML Extension (Version 2):

```
<style type="text/css">.textbox { [[ CSS Declarations ]] }</style>
  ...
<div class="textbox"> ... </div>
```

Borderline HTML Extension (depending on the `textbox.css` ***origin):***

```
<link rel="stylesheet" type="text/css" href="...textbox.css" />
  ...
<div class="textbox"> ... </div>
```

"Intensional" HTML Extension:

```
<textbox> ... </textbox>
```

Fig. 3. Extensional vs. Intensional HTML Extensions

point of view, Version 2 is just an abbreviation of Version 1, which enables the reuse of style information inside an HTML page.

The borderline version also uses the `class` mechanism, but the HTML class refers to an external CSS resource. As mentioned above, this can be regarded as extensional or intensional, depending on the author of the external CSS resource.

In the intensional case of the example in Figure 3, the new element `textbox` simply appears in the HTML (which should use a `DOCTYPE` declaration declaring the new version), and it is assumed that semantic information about the formatting of this new element can be found in the extended schema, or, in case of HTML and CSS, in an associated style sheet (generated from the schema or authored separately), where there can be CSS code associated with a `textbox` element type selector, containing the same code as the `.textbox` class selector of the extensional case. Today's browsers are not equipped to dynamically adapt to new HTML versions, but most hooks are there to implement such a behavior.

To summarize, HTML provides support for extensional as well as for intensional ways to declare the semantics of extensions. However, since there is no standardized way to associate an "extension formatting semantics CSS" with a new version of HTML (instances as well as schemas), all browsers available today do not support schema-based extensions; all that can be implemented through the browser mechanisms are instance-based extensions. The three main reasons for this lack of dynamism in modern browsers are (1) that HTML has remained stable for a long time now, (2) the lack of a standardized way of communicating non-declarative semantics, and (3) the browser authors' assumption that extensions of HTML probably not only require declarative extensions via CSS, but also processing semantics beyond the capabilities of CSS, which require a browser update to be fully supported.

4.2 Non-declarative Semantics

The approach of declarative semantics discussed in the previous section is to describe as much of the foreseeable semantics of extensions as possible. This makes it possible to make extensions semantically self-describing, at the price of having to implement the declarative semantics[9]. Generally speaking, declarative semantics in almost all cases will only cover a part of the full semantics intended for extensions, and these "full" or "true" semantics of extensions are called the *extension semantics*. In Figure 1, the declarative semantics (II) and the non-declarative semantics (III) together constitute the full extension semantics.

The most important question for the design and usage of declarative and non-declarative semantics is whether the mismatch between these two can be tolerated for a useful fraction of application scenarios. This semantics mismatch is discussed in more detail in Section 4.3. It should be noted that, in contrast to the purely declarative semantics, which by definition can be fully described through declarative mechanisms, the non-declarative semantics are often defined less formally, either by prose, or through procedural code performing certain functions.

In order to give an example for non-declarative semantics, Figure 4 shows an hypothetical extension of HTML with an element for marking up square root content, which should be formatted as a square root formula.

Assuming that the declarative semantics of the HTML extension is CSS and implemented intensionally, one possible way to describe the extension declaratively is to simply insert the text `sqrt(` before and the text `)` after the content of the `sqrt` element. This is not a particularly elegant way to render a square root, but is sufficient to convey the meaning of the content, and therefore probably acceptable.

However, the "full" semantics of the extension are to render the content as a square root, and the standard way to do this is to use a square root symbol ($\sqrt{}$) and to include the content under the top bar. Therefore, an extended client implementing the extension (or, in other words, implementing the non-declarative semantics of the extension) can format the square root using the appropriate mathematical symbol.

In this case, the difference between the declarative semantics and the non-declarative semantics is obvious. However, most users probably agree that rendering the square root as `sqrt()` is good enough to create an understandable

[9] In the case of CSS as described in the previous section, this price is quite high. CSS has become so complex that there is no single browser supporting the full CSS language repertoire.

HTML Extension for Square Root Element:

```
Square root rendering looks like <sqrt>2</sqrt>
```

Declarative Formatting Semantics (CSS):

```
sqrt:before { content : "sqrt(" }
sqrt:after  { content : ")" }
```

Formatting of HTML Extension:

Declarative Semantics:	Square root rendering looks like sqrt(2)
Non-declarative Semantics:	Square root rendering looks like $\sqrt{2}$

Fig. 4. Declarative vs. Non-declarative Semantics for HTML Formatting

rendering of the markup, so the difference in semantics is acceptable. This question is discussed more generally in Section 4.3. In this case it is also interesting to note that the non-declarative semantics effectively "overwrite" the declarative semantics. This happens frequently, but the overlap between declarative and non-declarative semantics varies and cannot be described generally.

As mentioned in the introduction to this section, semantics always depend on context. Consequently, non-declarative semantics often include new and unforeseeable context (such as new processing steps) that cannot be covered by the declarative semantics of a given extension framework. The interesting question always is whether it is possible to describe extensions sufficiently using the declarative semantics allowing a "graceful degradation" of the behavior as described in Section 2.

4.3 Semantics Mismatch

Starting from Figure 1, it can be asked whether the initial semantics of the application (I) are or at least can be described using declarative semantics. Usually, this is not the case, because of the unnecessary effort this requires, and also because the initial semantics often are more complex than the rather limited declarative semantics for extensions can describe.

Applications interpreting data with unknown extensions are limited to the declarative semantics (II) contained in the instance (extensional) or the extension schema (intensional). In most cases, the full extension semantics are a superset of the declarative semantics, also described by non-declarative semantics (III). The non-declarative semantics, however, are implemented only by applications having advance knowledge of the extension, so earlier versions of the application are limited to the declarative semantics. These limited semantics can have two consequences:

- *Graceful Degradation:* If the limited declarative semantics are sufficient to guarantee the correct functioning of the application, possibly with some compromises regarding the quality of extension processing, then it is reasonable to accept the limitation in semantics. The earlier example from Figure 4 of rendering a square root textually is an example of how the limitation of the declarative semantics is acceptable, because the limited extension processing still yields the intended result: a human-readable form of displaying the square root of an operand.
- *Extensibility Failure:* If it is considered inadequate to process an extension or particular instances of an extension with declarative semantics only, then this should be signaled to older applications by specifying this information, ideally as extensional information (i.e., as part of the instance data). The application behavior regarding extensibility failures can range from signaling an error to more graceful behavior such as recommending updating the application to enable better processing of this particular extension respectively its instance data.

  Moreover, extensibility failures can raise different error levels, and they can be specific to a type of declarative semantics, so that only applications interpreting these semantics are affected by the extensibility error. Consequently, extensibility failures should be more specific than a simple `mustUnderstand` flag, which is the simplest form of extensibility failure information. Most likely, extensibility failure information should include the error level, and the affected semantics.

Generally speaking, the question of how well the declarative semantics can handle the actual evolution of the application depend on the complexity of the declarative semantics, and of course the ability to design these semantics in advance. An interesting example is the ongoing evolution of CSS. CSS can be considered the declarative semantics (for formatting information) of HTML, and the specification has grown from a rather limited set of formatting instructions in CSS1 to the complex variety of formatting modules constituted by CSS3 [10]. The problem is that CSS has become so complex that no browser fully supports the complete CSS vocabulary, which is an indication that the costs of implementing the complex declarative semantics seem to outweigh the benefits.

During CSS evolution, new semantics were added to CSS, for example the ability to specify table formatting, which was not included in earlier versions. However, even CSS3 does not cover all areas of HTML or possible extensions to HTML, for example there is no support for hyperlinking or mathematical formulas. Nonetheless, CSS is an interesting example of an *extension of extension semantics.* The price for this is that older versions of browsers do only support earlier versions of CSS, so that they cannot interpret all CSS information that may be used in newer documents. In case of HTML/CSS, the standard mechanism is to ignore unknown instance data[10], but in the general case, the

[10] Ignoring unknown instance data is the required processing behavior for HTML elements and attributes, and for CSS selectors and declarations.

management of the initial schema, extensions to it, declarative semantics, and the extension of declarative semantics is a challenging task.

Again, it is a question of how much effort one would like to invest into the design and implementation of the extension framework, and how much benefit this effort and cost is going to generate. Because the *extension of extension semantics* introduces a new level of complexity, it is not discussed in this paper.

5 Extension Framework

The complete framework for semantically extensible schemas for Web Service evolution combines the aspects presented in previous sections with guidelines and rules to control the evolution and to mandate processing guidelines which must be observed by peers participating in the Web Service scenario. The framework includes:

– *Open and Extensible Schemas:* Without appropriate schema modeling, there can be no controlled evolution, so the design guidelines for openness and extensibility of schemas outlined in Section 3 must be followed.
– *Extension Semantics:* Since the goal is the semantic extensibility of Web Services, the rules for extension semantics as presented in Section 4 must be followed. Without extension semantics, extensions can only be ignored.
– *Extension Guidelines:* Since current Web Service technologies (SOAP, WSDL, and UDDI) do not provide any support for versioning, the versioning process must be controlled by the user. This should be done through a set of extension guidelines, which document all necessary steps and interdependencies that are required for defining, implementing, and deploying a new version of a Web Service.
– *Processing Rules:* Participating peers must follow certain guidelines when implementing extensible Web Services, in particular they may not ignore information that is required for proper processing of extensions. Since these rules go beyond basic Web Service semantics, they must be enforced by mandatory processing guidelines for participating peers.

Implementing this framework in a Web Service scenario is not trivial. In particular, the design of declarative extension semantics (possibly including multiple semantics for different application aspects) can be hard. Furthermore, the extension guidelines impose additional constraints on schema designers, and the processing rules impose additional constraints on implementors. It is thus necessary to carefully weigh the cost of the extension framework against the benefits of using it. Important factors in this analysis include:

– *Controllability of Software Versions:* If the application scenario is a closed environment where software updates can be controlled and can be realistically deployed within a reasonable time frame, then investing in software version management may be a better solution.

– *Server/Client Ratio:* The hard part in extensibility is forward extensibility, in case of Client/Server-Scenarios this means old clients with new servers. If the number of clients is small, client updates can be achieved much easier. If the number of servers is small, backward compatibility may not be an issue, because servers can easily be updated.
– *Frequency and Locality of Extensions:* Building extensibility into the software is only reasonable if the frequency of extensions is higher than the normal versioning cycle of the software. Frequent extensions, or extensions being introduced by different participants in the application scenario make the effort of implementing extensibility more likely to be beneficial.
– *Predictability of Extension Semantics:* They key of useful extension semantics is the ability to capture as much of the semantics of potential extensions as possible, because only then it is possible to design a declarative language for them. If the application scenario is likely to produce extensions with significant semantics that cannot be represented declaratively, then either the declarative extensibility approach is inappropriate, or it should be upgraded to an (even more complex) approach for extensible extension semantics.

Thus the framework for semantically extensible schemas for Web Service evolution is not the perfect solution for every Web Service scenario. However, there are many scenarios which may benefit from a managed way of Web Service versioning, and declarative extension semantics provide a particularly powerful way of managing extensions. The framework presented in this paper is not formally complete, and has evolved from concrete application scenarios rather than a purely top-down approach. We therefore think that further work is required in this area, and that more real-life applications are required to find the optimal balance between framework complexity and the average cost/benefit ratio.

6 Conclusions

The framework presented in this paper is the result of two real-life scenarios where extensibility was a key point of the requirements. However, before it can be turned into a completely defined formal way of designing semantically extensible Web Services, more examples are needed for evaluating the framework.

Also, some rather complex problems have only been discussed very shortly, such as the question of if and how the extensions of the schema (discussed in Section 3) and the associated semantics (Section 4) can and should be coordinated in a well-defined and robust way. Then there is the interesting issue of an evolution of the declarative semantics, so that there can be a need for an extension framework for these semantics (which thus addresses the issue of *extensible extensibility*).

Since wide-scale Web Service deployment still is a rather rare case (in most cases, Web Services are used in small and controlled settings), the question of Web Service versioning so far has not become a real problem. In the medium term, however, this will happen, and the key Web Services specifications (SOAP, WSDL, and UDDI) will need to provide better and more advanced answers than SOAP's current `mustUnderstand` attribute, the only means of versioning support in the current Web Service specifications. This paper gives no final and perfect answers, but presents approaches which provide a reasonable solution to the problem of Web Service evolution.

Once Web Services are deployed more widely and more loosely that today, Web Service evolution will become a serious problem. Current Web Service technology does not provide adequate support for this type of problem, and the framework for semantically extensible schemas for Web Service evolution presented in this paper augments the existing Web Service technologies to better support Web Service versioning.

References

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: Extensible Markup Language (XML) 1.0 (Third Edition). World Wide Web Consortium, Recommendation REC-xml-20040204 (2004)
2. Chinnici, R., Gudgin, M., Moreau, J.J., Schlimmer, J.C., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. World Wide Web Consortium, Working Draft WD-wsdl20-20040326 (2004)
3. Orchard, D., Walsh, N.: Versioning XML Languages. World Wide Web Consortium, Proposed TAG Finding (2003)
4. Su, X., Brasethvik, T., Hakkarainen, S.: Ontology Mapping through Analysis of Model Extension. In Eder, J., Welzer, T., eds.: Short Paper Proceedings of the 15th Conference on Advanced Information Systems Engineering. Volume 74 of CEUR Workshop Proceedings., Klagenfurt, Austria, Technical University of Aachen (RWTH) (2003) 101–104
5. Hunter, J.: MetaNet — A Metadata Term Thesaurus to Enable Semantic Interoperability Between Metadata Domains. Journal of Digital Information **1** (2001)
6. Castano, S., Ferrara, A., Kuruvilla Ottathycal, G.S., De Antonellis, V.: A Disciplined Approach for the Integration of Heterogeneous XML Datasources. In: Proceedings of the 13th International Workhop on Database and Expert Systems Applications (DEXA 2002), Aix-en-Provence, France, IEEE Computer Society Press (2002) 103–110
7. Fallside, D.C., Walmsley, P.: XML Schema Part 0: Primer Second Edition. World Wide Web Consortium, Proposed Edited Recommendation PER-xmlschema-0-20040318 (2004)
8. Bray, T., Hollander, D., Layman, A., Tobin, R.: Namespaces in XML 1.1. World Wide Web Consortium, Recommendation REC-xml-names11-20040204 (2004)
9. Ghezzi, C., Jazayeri, M.: Programming Language Concepts. 3rd edn. John Wiley & Sons, Chichester, England (1997)
10. Meyer, E.A., Bos, B.: CSS3 Introduction. World Wide Web Consortium, Working Draft WD-css3-roadmap-20010523 (2001)

A Profile Based Security Model for the Semantic Web

Juan Jim Tan and Stefan Poslad

Department of Electronic Engineering, Queen Mary, University of London
Mile End Road, London E1 4NS, UK
{juanjim.tan, stefan.poslad}@elec.qmul.ac.uk

Abstract. The trend towards ubiquitous public services is driving the deployment of large scale, heterogeneous, semantic distributed service infrastructures. The critical and valuable assets of open services need to be protected using heterogeneous security models such as multiple domain-specific authorisation and access control mechanisms. A dynamic approach to managing inter-domain security to support openness is required. A semantic model that uses profiles, that supports policy type constraints and that supports profile-based security information interchange for multi-domain services has been developed.

1 Introduction

Multi Agent Systems (MAS) are emerging to be deployed in multiple, heterogeneous domains especially in the area of multi service composition and planning such as dynamic supply-chains, opening up newer, more customised consumer markets. The security issues of such Multi Agent Multi Domains (MAMD) are more complex to manage and support than in single homogeneous application domains. More research is needed to support distributed heterogeneous services in which the management of security configurations can be discovered, orchestrated and enforced. Numerous initiatives from standards consortia such as IETF, W3C and OASIS and various market leaders, have developed more open models and specifications to enable distributed systems to securely interoperate. No single security standard is suitable for all application requirements and infrastructures. The choices in selecting and configuring the security can lead to a lack of interoperability. The lack of a holistic solution that harmonises the various security models is a major obstacle to the wide-scale deployment of open systems. An agent-driven policy security management model that supports the discovery and subsequent matching of security profiles is proposed. Profiles are defined in a model called V-SAT: Viewpoints of sets of Safeguards that protect the assets from threats [1] .

2 Related Work

Security for open systems faces many new challenges when operating either in closed homogeneous and heterogeneous domains. Whilst static security configurations can be specified and agreed in advance, for example Web-clients that utilise HTTPS to support confidential information, this lacks flexibility in a dynamic heterogeneous domain that has multiple security requirements. The current plethora of different active security standards indicates that no single security framework is suitable for use with heterogeneous distributed systems. This has lead to the use of mediating models that are able to bridge between disparate security standards [13]. Here, security mechanisms and objects are represented using DAML+OIL ontologies in order to allow agents to specify security requirements and capabilities. Service descriptions are published in DAML-S, but this does not yet address other emerging service models such as "*BPEL4WS+WSDL*" [14] and "*WSCI*" [15]. User and service requirements are paired, and can be negotiated if they do not match. A bridging method can be used to mediate between different security standards but there is no explicit conceptual model to support open heterogeneous security. Overly sophisticated negotiation of security requirements in open environments can result in complex systems that are too brittle and impractical for large scale interoperable deployments.

Table 1. Comparison between V-SAT, KAoS, Rei and Ponder

	V-SAT	KAoS	Rei	Ponder
Ontology based	Yes	Yes	Yes	No
Policy Type	Constraint or Pre-condition based	Access Control based	Access Control based	Access Control based
Policy Representation	KIF + RDF(-S) **developing support for other representations*	OWL	Rei: (Prolog-like syntax + RDF-S)	Ponder language specification
Open Security Interoperability Support	Security Profiling – capable of representing security instances and complex processes of open systems	Applying mediating and proxy agents onto specific domains	Applicable in specific domains	Applicable in specific domains
Reasoning Support	Java Theorem Prover	Java Theorem Prover	Prolog engine	Event calculus representation

The management of large, multi-domain distributed systems is often itself distributed to support scalability. Management domains provide the means for partitioning management responsibility by *grouping* objects in order to specify policies including access control policies. *Access control policies* specify the operations a group of *subject* objects is permitted to perform on a group of *target* objects. Research has focuses on policy specification languages such as Ponder [2], TPL [11], and PDL [20] that define rules for policy management. Several of these policy-based management technologies are significant and have developed algorithms to handle conflict and resolution, such as the KAoS service [12]. As the scope of these policies is specified in terms of management domains - access control decisions are mainly based on the authenticated domain membership of the subject and can be difficult to apply towards open service environments where membership is often more unpredictable. In Table 1, we define some comparisons between our V-SAT model against other leading policy based distributed systems such as KAoS [12], Rei [16], and Ponder [2].

In Table 2, we define some comparisons between our semantic based V-SAT model against the WS-Security Specifications [21] and Grid Security [22] initiatives.

Table 2. Comparison between V-SAT, WS-Security, and GRID Security

	V-SAT	WS-Security	GRID Security
Ontology based	Yes – Semantic	No – Syntactic	No
Policy Support	Constraint-based	Constraint-based	Authorisations
Policy Representation	KIF + RDF(-S)	XML	Grid policy / WS-Security
Interoperability Support	Yes – Security Profiles	Security Token – Claims	GRID environment / WS-Security Spec.
Reasoning Support	Java Theorem Prover	Not defined	Not defined
Risk Management	Yes	No	No
Analysability	Ontology simplifies reasoning and dependency validation	Specific to a particular context / domain and lacking in inter-concept validation	
Expressivity	Represent behaviour of complex environment, multi level abstractions, and easy to extend new concepts	Represent only specific behaviours and difficult to extend new concepts	

Syntactical specifications such as WS-Security support the representation of security information using other XML initiatives such as OASIS and W3C. Grid security on the other hand, is based on specific Grid based policy representations currently support the WS-Security specifications. The V-SAT model utilises a semantic based approach to model security representations within open service environments. This approach adds value to existing security models in two ways, in the first instance it harmonises technologies such as WS-Security, Grid, SAML, etc, hence advocating interoperability amongst disparate systems using a common ontology. In the second, it can support the adaptive management of open systems using risk and reasoning based models to detect Byzantine nodes and promote safer service environments. The use of semantics supports the expressivity and analysability of security information where an upper ontology can be shared and easily extended. Adaptive management models use policies to support dynamic security reconfiguration by managing systems at a management level without affecting the underlying implementations.

The concept of an ontology covers a range of models ranging from flat sequential syntactical models as advocated by the IETF, W3C byte stream protocols, hierarchical syntactical models (XML) and dictionaries of security terms, to more expressive logic based frameworks based on RDF and DAML.

Table 3. Summary of Existing Solutions

Functionalities Specification	Confidentiality	Authentication	Integrity	Authorisation	Non-repudiation	Credential Management	Interaction Protocols	Policies	Trust Delegation	Semantic Service Adaptable
XML Encryption	✓		✓							✓
XML Signature		✓	✓		✓					✓
XACML		✓	✓	✓	✓			✓		✓
SAML		✓	✓	✓			✓			✓
XKMS			✓			✓	✓			✓
SESAME	✓	✓	✓	✓						
PICS			✓	✓	✓			✓		
P3P								✓		✓
KeyNote			✓	✓		✓		✓	✓	
S/MIME / PGP	✓	✓	✓							

Table 3 contrasts some common security standards. There is a raft of issues in developing open models to support MAMD systems. The need to integrate the various solutions to enable the engineering of integrated management systems has already been acknowledged [3] and, in recent years, emphasis was placed on policy specification [4] and information models for managed objects including security [2]. But there has been little focus on approaches for supporting the management of MAMD environments containing heterogeneous entities. We believe that a common high-level model for grounding various entities of MAMD systems related to security is important.

3 Requirement Analysis

Architectures for enforcing policies are moving towards strongly distributed paradigms, using technologies such as mobile code, distributed objects, intelligent agents and programmable networks. Borrowing the terminology from Flatin et al. [5], paradigms in which the management task is delegated to distributed entities that actively participate in the management decision making are called *strongly distributed*. These entities interpret and enforce policies, and their behaviour is dynamically changed by those policies. In contrast, *weakly distributed* paradigms are those in which the management decision-making is concentrated in a few managers, with distributed agents or entities acting merely as data collectors. In this paper we aim to design a *strongly distributed* management and the following is a list of requirements that the model specification should support: explicit representations of security profiles; flexible and distributed management of security profiles situated in a heterogeneous environment and the use of profiles that specify constraints for policies adhering to these profiles.

4 Layered Security Framework

The development of a holistic ontology is not an end itself: it provides the means by which security services and software such as agents can advertise and exchange security related information between them and it acts as a model for the management of security processes and services. An ontological model promotes the interoperability between disparate security systems, providing a framework to support secure transactions across heterogeneous multi-domain boundaries. Developing an ontological model has raised many issues regarding the modelling choices taken in development of knowledge representation models. There exist numerous ways of representing security specification information ranging from informal models to formalised specifications that provide epistemic descriptions of knowledge. Therefore, a practical and normative way of representing information is required. The main aim of utilising a representation model in our context was mainly driven by the following motivation [18]:

- Knowledge should represent the basis for intelligent reasoning, where facts can be compiled into logical descriptions that are applicable in a domain.
- An expressive language that is able to facilitate vivid representation of knowledge is needed. This enables users to define realistic models of their intended domain to aid stronger comprehension and communication.
- Besides having a well defined representation language, the information must be encoded in a normative form that allows efficient computation. This aids in practical systems that are able to collectively gather knowledge for processing.

In our model, profiles describe general relationships between safeguards, assets, and threats. Particular relationships are defined as rules or constraints called policies. In policy models, management is supported through the specification of policies as rules that act as qualifiers for the action relationships of subjects on targets. For example, a message target (asset) uses confidentiality actions such as encryption to protect against disclosure to general agent subjects. Policies define the conditions when encryption is used. The subject-action-target type policy models express directed asymmetrical actions from subjects to targets. Although our conceptual model can express such subject-action-subject relationships and policies, it is also easy to express other types of relationships such as explicitly defining heterogeneous bidirectional and multi-party security interactions between multiple senders and receivers. For example, senders may be required to sign messages, mediators may be required to verify signatures on behalf of receivers and receivers may use access control to receive messages.

We use an ontology representation for the profile-based security model. Ontologies in DAML+OIL or OWL support the normative encodings and can express logics for intelligent reasoning. The security ontology is specified using two approaches: an abstract approach for capturing stakeholders of the security environment and an explicit approach for mediating between security specifications. Security can be integrated with existing semantic based applications through high level conceptualisations and can be mapped to existing security specifications. In order to make the model more maintainable, there is a separation of concerns.

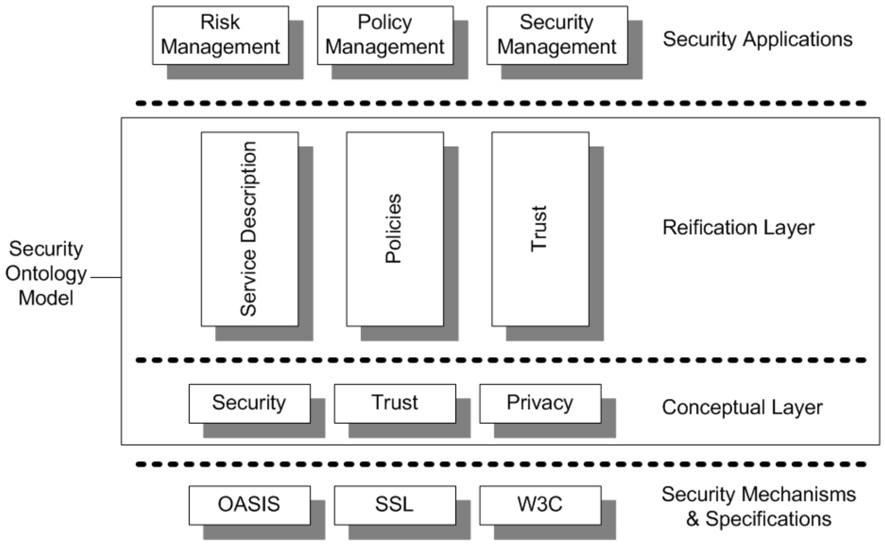


Fig. 1. Layered Ontological Security Model

The conceptual model is separated from different commitments to use those concepts such as service advertiser and policy-based management applications. The ontological model (see Fig. 1) consists of a:

- *Conceptual layer*: defines the properties and relations between security, trust and privacy related concepts;
- *Reification Layer*: comprises the following sub-layers:
  - *Service Description Layer*: the means for security processes to be hooked into service processes.
  - *Policy Layer*: provides the means for defining security rules and constraints.
  - *Trust Layer*: provides the means for defining trust implementations within systems to enable soft security interoperability between disparate applications. This is however not the focus of this paper and is not discussed further.

These are separated from:

- *Security Mechanisms*: specific instances of security concepts, policies and service entities as defined in existing security standards;
- *Security Applications*: that makes commitments to use the security ontology within specific application domains.

This separation enables the security conceptual model to be made independent of the application requirements and the use of specific security mechanisms. In the following sections the ontological model is defined in more detail.

4.1 Security Profile

The Security Management Model is concerned with relationships between safeguards, assets and threats entities. The profile defines sets of relationships such as how particular safeguards protect particular assets against particular threats.

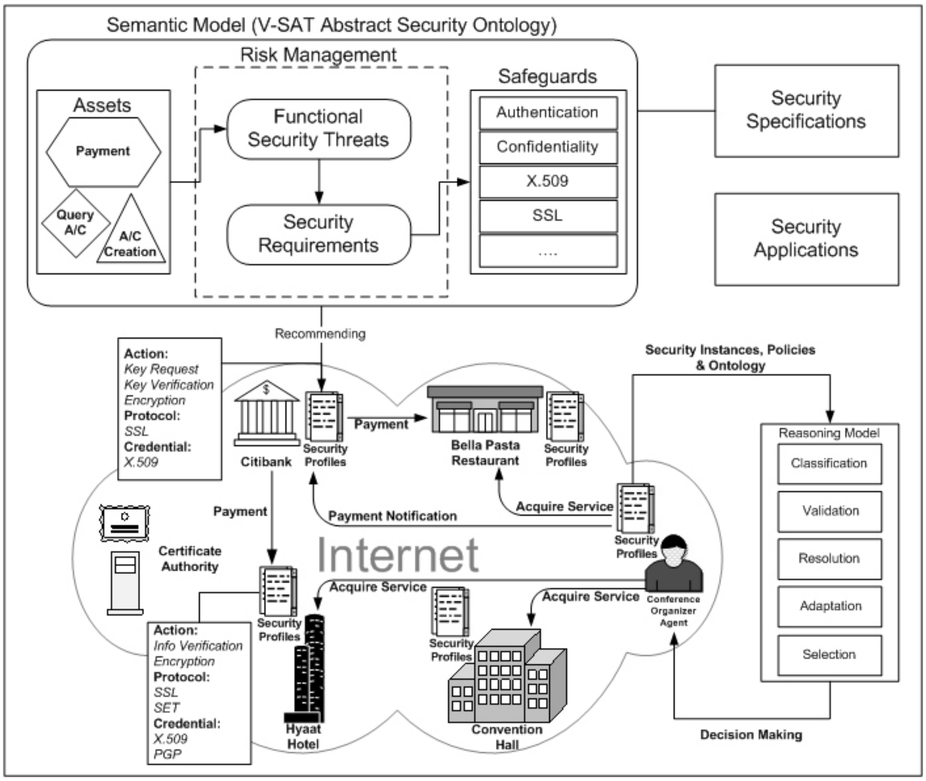


Fig. 2. Profile Driven Security Scenario

To make these ideas more comprehensible and appraisable, a scenario is given in Fig. 2. The example describes an open environment setting where different systems publish their services along with them externally public security configuration and requirements. These service descriptions have included a detailed service process description of their security choreography or workflow. The security processes are represented as profiles enforcing their respective security configurations (protocols, credentials, and actions) and policies. Some of these configurations include combinations such as SET or SSL, X.509 certificates, and "Confidentiality" or "Authentication" actions. In the scenario, A 'Conference Organizer Agent' wishes to organise an event and interacts with services such as a convention hall, a restaurant and a hotel. Initially the agent discovers these services through directories and discerns the security choreography and profile. Through the profile description and policies governing these security instances, the agent is able to reason about the profiles with the aid of the holistic security ontology, hence capturing and understanding the concepts and processes needed to support interoperability between disparate services. Eventually payment is made through the banking service and a conference event is organised. The security profiles represented by each service are created by the security ontology and risk models. Agents interacting in this environment would utilise security applications and the reasoning model for acquiring various security configurations. The concept of profiles for supporting security within a heterogeneous service environment is explained below.

4.1.1 Singular Profile

A profile exists to support binding relationships between arbitrary entities within a MAMD environment. These profiles can collectively provide a complete security description or viewpoint of a service to support dynamic security management [6]. Profiles are versatile in managing the switching between active and inactive behaviour of policies to ensure controllable environments in MAS. Asymmetrical relationships shared between assets, safeguards and threats are put together as a service profile with their accompanying policies. A profile can express various policy rules, defining the security instantiations and pre-conditions supported. This represents a meta description of the services' security workflow and service descriptions to promote the discovery and interoperability of services from separate domains.

4.1.2 Composite Profiles

Composite profiles are defined as a collection of one or more safeguard, asset, and threat entities belonging to an open service. The composite entities can be seen as a single managed object belonging to a service or domain. The collection of entities can be encapsulated within a composite profile that holds the internal relationships of service processes and policies. A profile can be part of one or more profiles comprising a collection of profiles managing the complete security workflow of a system. This model constitutes the design of the MAMD architecture where various levels of composite and overlapping domains exist in a variety of heterogeneous topologies creating composite profiles for managing service entities.

4.1.3 Security Configuration Policy Entities

MAMD security management involves monitoring the activity of a system, planning relationships between entities, describing policy rules, enforcing management decisions, and instantiating relevant service control actions to manage the behaviour of the system. The policy service provides the interpretation of policies within profiles in order to achieve the overall objectives of the communicating assets, and consequently supports risk monitoring of the prevailing threats that can be iteratively fed into the existing system.

The concept of policy and profiles is such that whereas the former influences the pre-conditions of the *instance*, the latter specifies the relationship between *subject* and *target instances*.

In MAMD systems, the number of assets could be large, rendering it impractical to specify policies in terms of individual assets - it is easier to group assets into domains where a policy applies [8]. However, entities within a MAS environment are autonomous components or services that are complex, and may need specific policy management support from the state of the world they interacts with. Therefore to optimise the number of policy objects into a manageable scope, the use of an explicit ontological model provides the relationship mappings of security concepts and their specifications as profiles to permit the clustering of similar safeguards and policies into discoverable entries [6].

According to [7], an arbitrary predicate based on the value of an object attribute can be used to select sets of subject or target objects to which a policy applies. A search over all reachable objects, within a distributed system, to determine these sets is clearly impractical. The selection of objects is thus limited to be within a domain, whose membership is known. However, the use of appropriate security ontologies, along with policies that specify the necessary pre-conditions for membership and domain information can enhance the management of distributed profiles in multiple domains.

Interacting services and agents may result in conflicting profiles because their policy constraints do not match. In many cases, it is not possible to prevent conflicts, so it is necessary to detect and resolve them. The matching of policies within the reasoner is a daunting task. The idea of supporting policy negotiation for conflicting policies is potentially non-scalable and can introduce high overheads and complexity. Therefore, to enable alternative solutions for policy conflict resolution, the introduction of alternative policies (*n-arity*) may be defined, e.g., in Fig. 3 an optional 'Key-Length' can be supported.

```
<profile:KeyInfo rdf:ID="keyInformation">
 <policy rdf:range="xsd:String">
     ( exists (?a (KeyLength ?a)) (or (= ?a 1024) (= ?a 512) )
 </policy>
</profile:KeyInfo>
```

Fig. 3. Example of an *n-arity* Policy

The above policy defines the key bit size length of a particular credential, where a disjunction operator supports either 1024 bits *or* 512 bits key lengths. The order characterises the preferential precedence of the system and can also be used for resolving policy conflicts. Consequently, security profiles can be executed by the security application layer in which, policy, privacy and cryptographic computation management can be enforced.

A **security configuration policy** specifies the conditions of safeguards descriptions or security requirements that external entities should abide by when utilising services offered by the system. Profiles occasionally require assets to adhere to certain conditions or rules. Hence the use of DAML-S pre-conditions is needed to offer a more precise formulation of expressions. However, the current DAML-S pre-condition descriptions for specifying these input rules are vague and do not provide examples for integrating services.

As a result, other constructs are used to express these conditions. These constructs are based upon introducing a string-based property in relation to a DAML-S precondition. Using a string based property has its advantages; it provides the freedom for specifying the required condition of the process, and its proprietary representation. However, the use of structured semantic representation methods is encouraged to provide a more normative way of representing policies. This allows policies to be specified using a multitude of languages. Having highly configurable systems can easily lead to bad configurations, thus the support for heterogeneity would benefit from defining policies based on a commonly agreed security ontology.

The profile based approach has been applied to service discovery technologies such as *"DAML-S"* and *"BPEL4WS + WSDL"* (available at [10]). Having developed profile description examples for both web and agent services in [6], we can further implement our results into other emerging technologies such as *ebXML*, *RosettaNet* and *WSCI*. These grounding examples provide the basis for advertising security configurations of heterogeneous web and agent services.

4.2 Security Management Applications

Management applications are constructed as a set of cooperating components that can be aggregated to offer more complex management support. There may be enforcement mechanisms such as agents and multiple enforcement proxies that act on behalf of the application to perform these activities. Management applications can support:

- Distributed management of service security discovery, matching and reasoning about profiles.
- Wrapping of the independent underlying security mechanisms to support confidentiality, non-repudiation, authentication and key management.
- Enforcing the operation of mechanisms based on the specified policy security requirements.

5 Evaluation and Discussion

The profile-based security model has been specified and implemented in demonstrations of services, such as market places, event organisers and e-banking agent systems, as part of the EU funded Agentcities.RTD project [17]. An explicit ontology defining abstract concepts that can be specifically grounded with normative security specifications, such as SAML, XKMS, XML Signature and XML Encryption was specified and implemented. The ontology provides Viewpoints of associations between safeguards, assets, and threats (V-SAT) defined using profiles [1]. The ontology is available at http://agents.elec.qmul.ac.uk/agentcities/security/ontology.htm. We present, in Fig. 4, part of a directory entry describing an advertised service with references to the core and functional ontologies in FIPA based service application.

An application has been built using JTP (Java Theorem Prover) and a security API. The API can be triggered to support authentication, confidentiality, integrity, key management and key distribution services after high level processing such as discovery, reasoning and orchestration has been established.

Multiple service profiles can be registered, for example, a ‘Secure Tunnelling’ service can have the following sub-services (actions) such as ‘Authentication’ and ‘Key Exchange,’ where each action represents a security profile. The use of multiple profiles is supported using ontology extensibility and helps to maintain a loosely coupled relationship between profiles. As a result, classifying actions into profiles allows us to manage them collectively.

```
(df-agent-description
....
 :services (set (service-description
   :name Authentication :type Security
  :ontologies (set http://agents.elec.qmul.ac.uk/agentcities/security/AbstractSecurityOntology)
   :properties (set (property
     :name VerifyCredentials
     :value http://agents.elec.qmul.ac.uk/agentcities/security/VFCredentialProfile))))
.....)
```

Fig. 4. A service advertisement showing how the interlinked use of the security ontology

The profile-based framework uses an abstract security knowledge model where conceptual representations of security entities are mapped onto explicit security specifications [1], [6]. This combines *generality* and *practicality* for expressing high level security relationships of application scenarios that can also be explicitly defined. A set of core functional security requirements such as authentication, integrity, and confidentiality is defined as core safeguards of the model. Further configurability is specified within actions, protocols, and credentials. The model also advocates extensibility, where specific safeguards can be extended or incorporated into the ontology.

Security profiles representing the process, configuration and instances of the system can be constrained using specific policies. An example policy constraining the algorithm of the retrieval method in a signature is given in

Fig. 5.

```
<policy rdf:range="xsd:String">
   (exists (?a (Transform ?a)) (or (Algorithm ?a  http://www.w3.org/2000/09/xmldsig#rsa-sha1)
    (Algorithm ?a http://www.w3.org/2000/09/xmldsig#dsa-sha1) )
</policy>
```

Fig. 5. An example policy for signature verification

The specification provides the normative descriptions for specifying policies within this framework. The language is modelled at an instance value level as opposed to a conceptual level. As a result we are able to produce more specific and flexible representations for specifying policies expressively, but with the disadvantage of introducing a large number of syntaxes.

5.1 Modelling Choice

The policy model supports reconfiguration of security requirements and conflict resolution using *n-arity* policies, and can be compared against other models such as [12], [2] and [16]. These models are similar in the nature to policy management infrastructures that support access control mechanisms for domain based environments. Multi domain interoperability between disparate open architectures can be incorporated [12], but the notion of open is constrained to membership domains where entities must be registered in advance. Policy models [16] that define rules to support access control for domain based security in RDF for the semantic web are complementary to our model. The open MAMD model can be extended, in which segments or collections of domains can be implemented using policy management technologies expressed in [12] and [2]. But, our system provides a holistic model for dynamically specifying security configurations between open systems to promote interoperability.

The abstract security ontology coupled with security specifications provides the model with the means for sharing knowledge between disparate services using a Knowledge Base. There is a trade-of in analysing MAMD system security in such an abstract way. The advantages of this kind of abstract reference model includes being insulated from popular particular technological security models that may become disused or frequently superseded and being able to support heterogeneous application security requirements. The disadvantage is that an abstract model may appear to be too abstract, complex and flexible to be used to specify concrete MAMD security systems for particular application requirements in practice. In order to minimise the disadvantages, a profile-based approach is used to map an abstract common view of security to particular application-oriented reifications of the model.

Having highly configurable systems can easily lead to bad configurations, thus the support for heterogeneity would need to follow an agreed semantics. Policies based on semantics of a security ontology, where the ontology provides a limited set of nouns to associate its facts as rules in policies, can provide additional support to cluster policies and so optimise management.

5.2 Viewpoints, Profiles, Contexts, and Its Semantics

The security ontology model and its semantics define concepts. Unfortunately, knowledge can be interpreted in different ways to make different sense from single or many disjoint situations. Subsequently, these situations can either be conflicting or contradictory to actually not make any sense at all. Therefore the concept of viewpoints (composite profiles), profiles (scenarios or situations), and contexts (meaning of the profile) provides an account for distinguishing the semantics of representing various similar instances of the common knowledge at different granularities. Fig. 6 and its attached formula explain a normative example on the concept of viewpoints, profiles and contexts to resolve its semantic anomalies.

In our definition, the security ontology represents a possible world model describing a wide range of stakeholders and entities. This representation expressed in ontology languages such as DAML+OIL is supported by model theoretic semantics describing a formal account of the interpretations of legitimate expressions of the language. Therefore, the profile represented in Fig. 6 defines the entities that are excerpted from the security world model, and that each collection of entities retains its identity over a period of time. These collections of entities could in this model be termed as a context, where it brings together a collective view of a situation that is an instance of the world model to resolve a security requirement of a service. In the conceptual graph, in Fig. 6, safeguards are the protection relationships between subject assets and threat targets. A graph also constitutes collections (profiles) of asset, safeguard and threat relationships representing an application context for an agent actor.

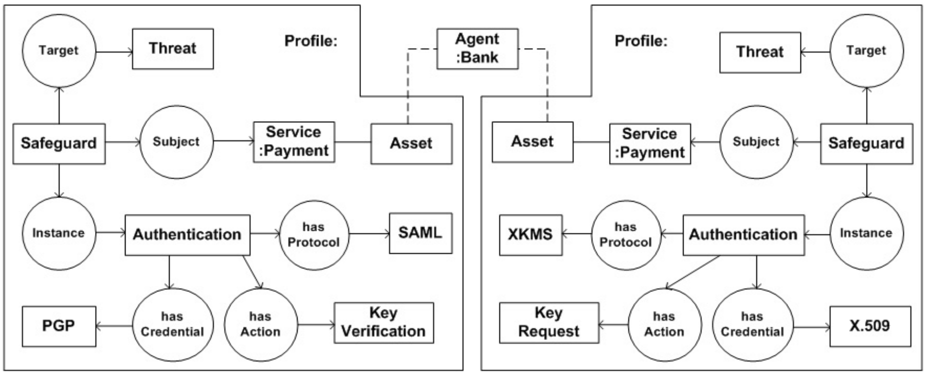


Fig. 6. Conceptual Graph of Profiles Constituting Contexts

$(\exists x\text{: Agent})\ (\text{identity}(x, \text{Bank}) \wedge$
$(\exists p\text{: Profile})$
$\quad (\text{dscr}(p, (\exists a\text{: Asset})\ (\exists w\text{: Service})\ (\exists s\text{: Safeguard})\ (\exists t\text{: Threat})\ (\exists y\text{: Authentication})\ (\exists f\text{: Protocol})$
$\quad (\exists c\text{: Credential})\ (\exists d\text{: Action}) \wedge \text{subClassOf}(w,a) \wedge \text{subject}(s, w) \wedge \text{target}(s, t) \wedge \text{instance}(s, y) \wedge$
$\quad \text{hasProtocol}(y, f) \wedge \text{has Action}(y, d) \wedge \text{hasCredential}(y, c) \wedge \text{asset}(x) \wedge \text{service(Payment)} \wedge \text{credential(PGP)} \wedge \text{action(KeyVerification)} \wedge \text{protocol(SAML)})) \wedge$
$(\exists z\text{: Profile})$
$\quad (\text{dscr}(z, (\exists a\text{: Asset})\ (\exists w\text{: Service})\ (\exists s\text{: Safeguard})\ (\exists t\text{: Threat})\ (\exists y\text{: Authentication})\ (\exists f\text{: Protocol})$
$\quad (\exists c\text{: Credential})\ (\exists d\text{: Action}) \wedge \text{subClassOf}(w,a) \wedge \text{subject}(s, w) \wedge \text{target}(s, t) \wedge \text{instance}(s, y) \wedge$
$\quad \text{hasProtocol}(y, f) \wedge \text{has Action}(y, d) \wedge \text{hasCredential}(y, c) \wedge \text{asset}(x) \wedge \text{service(Payment)} \wedge \text{credential(X.509)} \wedge \text{action(KeyRequest)} \wedge \text{protocol(XKMS)})))$

Fig. 7. Security Profile Formula of an Agent

The profile provides a rational bond between each disjointed entity within the security ontology. By modelling profiles as security identities of agents could result in a contradiction because each agent retains that identity over a period of time. In this case, an agent could result in having many profiles that may become contradictory to one another and may result in conflicts [19]. To avoid this contradiction, each profile can be represented as distinct contexts. To do this, profiles are considered meta but explicitly distinguished to define the contexts of each agent. These contexts can also be time-stamped to separate certain timelines they need to adhere to. In Fig. 6, we define an agent represented using multi profiles derived from the security ontology as contexts. The entities describing the context are distinct against other contexts and could be used conjunctively or disjunctively.*"*

The two profiles described by the propositions in Fig. 7 are nested inside the *dscr* predicates, and effectively define the explicit contexts. The descriptions inside those contexts refer to the agent *x*, which is quantified outside; but neither of the nested contexts can refer to or contradict any information in other contexts. Consequently, various contexts represent the different semantic meanings defining distinct security situations. These situations weave together sets of events to create collateral security viewpoints of the system. Viewpoints are a collection of many disjointed profiles and contexts that shares a consistent rational binding between one another. Hence, view-points provide the meta-representation of composite profiles that describes the complete security operations of systems or multi systems in an MAMD environment.

6 Conclusion and Future Work

MAS applications are increasingly being oriented towards heterogeneous environments to support business applications such as multi service composition and planning. Security issues arising from these multi-domain systems can be difficult to manage and support, especially with applications deployed using different security standards. Little work has been done to explain how the management of security can be enforced at different levels of abstractions. Hence, a layered holistic ontological model has been proposed to harmonise various security specifications. A concrete ontology has been created as the result of these investigations and the model has been applied using security profiles in an open environment. The policy layer is supported using a reasoning model to support the dynamic reconfiguration of security policies together with an operational model, making up the complete framework.

In the future, we plan to improve this model to support other industry standard representation languages such as *WSCI* and *ebXML*. In addition, there are plans to include trust description models and concepts to support interoperability between heterogeneous secure and trusted domains. As a result, open services can benefit from policy based environments that are dynamic and manageable through consistent use of viewpoints or profiles.

Acknowledgments. The research described in this paper is partly supported by the EU Agentcities.RTD (IST-2000-28385) and iTrust (IST-2001-34910) projects.

References

[1] Tan JJ, Poslad S, Titkov L, An Ontological Approach to Harmonising Security Models for Open Services. AT2AI, April 2004, Vienna, Austria.

[2] Damianou, N. C. (2002). Policy Framework for the Management of Distributed Systems. PhD Thesis, Imperial College. London, U. K., February 2002.

[3] Stone, G. N., B. Lundy and G. G. Xie (2001). Network Policy Languages: A Survey and a New Approach. IEEE Network, vol. 15(1), pp. 10-21, Jan. 2001.

[4] Hegering, H.-G., S. Abeck and B. Neumair (1999). Integrated Management of Network Systems: Concepts, Architectures and Their Operational Application, Morgan Kaufmann Publishers, August 1999.

[5] Martin-Flatin, J.-P., S. Znaty and J.-P. Hubaux (1999). *A Survey of Distributed Enterprise Network and Systems Management Paradigms*. Journal of Network and Systems Management, vol. 7(1), pp. 9-26, September 1999.

[6] Poslad S., Tan JJ, Titkov L., Agentcities.RTD D3.4: An Ontological Approach to Harmonising Security Models, June 2003. http://www.agentcities.org/.

[7] Sloman, M., Magee, J., Twidle, K., Kramer, J., An Architecture For Managing Distributed Systems. Proceedings of Fourth IEEE Workshop on Future Trends of Distributed Computing Systems, Sep. 22-24, 1993, Lisbon, Portugal.

[8] Morris Sloman, Policy Driven Management for Distributed Systems. Journal of Network and Systems Management, Plenum Press, Vol. 2 No. 4, 1994.

[9] Pennsylvania CIS Dept., January 2001.

[10] Agent Driven Policy Management for Securing Open Services. http://agents.elec.qmul.ac.uk/agentcities/security/

[11] Herzberg, A. et al. "Access control meets public key infrastructure, or: Assigning roles to strangers". IEEE Symposium on Security and Privacy, May, 2000.

[12] Uszok, A. et al. "KAoS policy and domain services: toward a description-logic approach to policy representation, deconfliction and enforcement". http://www.wiwiss.fu-berlin.de/suhl/bizer/SWTSGuide/

[13] Denker, G., Kagal, L., Finin, T., Paolucci, M., and Sycara, K. Security for DAML Web Services: Annotation and Matchmaking. ISWC, Sanibel Island, Florida, USA, 20-23 October, 2003

[14] BPEL4WS+WSDL, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[15] WSCI, http://www.w3.org/TR/wsci/

[16] Lalana Kagal, Tim Finin and Anupam Joshi. A Policy Based Approach to Security for the Semantic Web. ISWC, Sanibel Island, Florida, USA, October, 2003.

[17] Agentcities.RTD: Global Agent Testbed, http://www.agentcities.org/

[18] Davis, R. et al., "What is a knowledge representation?" AI Magazine, 14:1, 1993.

[19] Sowa, J.F. "Knowledge Representation – Logical, Computational, and Philosophical Foundations," Thomson Learning, 2000.

[20] Lobo, J., Bhatia, R., and Naqvi, S. "A policy description language". Proc. of AAAI, Orlando, FL, July, 1999.

[21] Web Services – Security, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[22] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke., A Security Architecture for Computational Grids, *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pp. 83-92, 1998.

A Framework for Authenticated Web Services

Carlo Blundo and Stelvio Cimato

Dipartimento di Informatica ed Applicazioni
Universita di Salerno,
84081 Baronissi (SA), Italy
{blundo,cimato}@dia.unisa.it

Abstract. New security challenges are arising from the widespread adoption of Web service technology. Indeed such technology does not directly address the problem of securing the information flow between the clients and the service providers. For this reason, many boards and technical committees are currently involved in research projects aimed to the definition of standards and specifications related to the provision of security properties. In this work, we propose a framework to accurately measure the number of accesses to a Web service, which can be invoked by authenticated clients only. The system we propose can be helpful in the development of new business models, where an *audit agency* is in charge of registering *clients* accessing the services, and of paying the *servers* which show a valid proof of the number of requests serviced.

Keywords: Web Service Security, Authentication, Metering, Cryptography.

1 Introduction

Recently, a growing interest is focusing on the development and deployment of Web Services. Many companies are investing large amount of money in research and development projects related to the exploitation of such kind of technologies. Indeed, Web services are usually presented as the next generation of distributed computing, which builds on and extends the well-known client-server model [17]. The "loose coupling" paradigm on which Web services technology is based, allows the development of applications which are interoperable, platform independent and discoverable.

The expected benefits from the adoption of the Web service technology involve both economical and technical aspects. The standardization and the flexibility introduced by Web services in the development of new applications translate into increased productivity and gained efficiency. Development costs are reduced by the use of standard interfaces which enable the integration among applications and the reuse of existing solutions, shortening the total development cycle.

Despite all those potential benefits, some obstacles hinder a widespread diffusion of Web services. According to several reports and industry observers, security concerns are one of the critical issues which should be addressed in order to foster a wider adoption of Web services [5]. Indeed, the original specification of the SOAP protocol, which is the lightweight messaging protocol allowing cross-platform interapplication communication, does not provide any security mechanism. Basically, Web service transactions are unencrypted and unsecured. So if from one side Web services developers open up networks allowing external users to access internal applications and data, on the other side administrators are presented with the problem of controlling and securing the information flow.

To find a remedy to such situation, standards efforts promoted by organizations (such as W3C, IETF, OASIS) have produced a set of specifications and recommendations composing a framework for the development of secure XML-based applications. Currently, many of these XML security initiatives are on the way to be completed in the next months. As an example, the Web Services Security Specification, proposing several SOAP extensions to integrate security guarantees for the exchange of XML messages, has been firstly published by IBM and Verisign in 2002 and accepted at the beginning of 2004 by the Organization for the Advancement of Structured Information Standards as an official standard. A widespread adoption of the standards defining security extensions for Web services is expected to start in the next years. Actually, the standard way to implement secure mechanisms for a Web service is the customization of SOAP messages, usually done by modifying the header of a SOAP request [4].

As soon as companies provide business services and release new products and tools Web services enabled, a problem which in our opinion will become more and more challenging, is the need of methodologies to accurately measure the number of accesses to a given service. Metering techniques have been developed in the context of measuring accesses to Web pages, where it is important to have a statistics on the exposure of advertising [1,8]. In the context of Web services usage, such metering techniques can be helpful in the development of new business models, where an *audit agency* is in charge of registering *clients* accessing the services, and of paying the *servers* claiming for payment, after that a valid proof of the number of serviced requests is presented.

The solution we propose is based on the well known idea of *hash chains*, previously used for user authentication and micro-payment systems [6,16]. The idea is to modify the header of a SOAP message in order to attach to a remote service invocation the data needed to authenticate the clients. Requests directed towards restricted services are filtered such that only clients holding a valid authentication token will be allowed to access the service.

The paper is organized as follows. In the next section, we give an overview on the emerging standard and techniques for securing Web service. In Section 3 we recall the metering scheme on which the framework we propose is based. In Section 4 we describe the architecture of the framework for authenticated access to Web service we propose, describing a prototype implementation in Section 5. Finally, we evaluate our solution with respect to the new standards (released after our work) and present some conclusions.

2 Web Service Security

Web services allow applications to exchange information overcoming the security boundaries usually defined by firewalls. Web Services need a transport protocol, such as HTTP, SMTP, FTP. In many cases, SOAP messages are transported over HTTP requests, which are usually unfiltered. A Web service enabled application could then expose several critical functionalities accessible trough the port 80 (or 443), which could originate security breaches if improperly or maliciously invoked. In a message based interaction it would be important to provide security guarantees on the origin of a message (*authentication*), the destination of the message (*authorization*), the protection of the content of the message from modifications (*integrity*) and unauthorized access (*confidentiality*) [2]. Unfortunately, none of the above properties are directly addressed by the original SOAP specification.

Traditional security techniques, such as standard SSL encryption over HTTPS, provide connection security and data privacy between service requestors and service providers. However, point to point security is not enough, since the service provider could be an intermediate destination, which could distribute pieces of sensible data to other providers.

Many technical committees and standard bodies are involved in the definition of standards and technical specifications related to the merging of traditional security technologies with the Web services and SOAP message definitions. The recently released OASIS standard regarding the security of SOAP messages, usually referred to as Web Services Security [12], defines new SOAP extensions, modifying the structure of SOAP message headers, and standard ways to attach security data to XML messages, in order to support different kinds of security mechanisms. In particular, such specification defines the use of XML signature [18] to provide SOAP message integrity, the use of XML encryption [19] to provide message confidentiality, and the ability to attach several kinds of tokens (e.g., Kerberos tickets, X.509 certificates, and so on) conveying security information to standard SOAP messages. XML encryption and XML signature are the specifications aimed to provide standard ways to include encrypted data and digital signature in XML messages. As discussed in Section 6, such mechanisms can be used in our architecture to carry the needed information to execute the protocol we propose.

Authentication and authorization information can be also expressed using the Simple Assertion Markup Language developed (SAML) by OASIS [9]. In SAML security information are contained in *assertions* which an issuer can state about a subject. Assertions describe the results of access control operations previously performed by some authoritative entity. SAML works via a centralized authority or in a distributed setting between group of trusting participants. SAML does not provide method to guarantee confidentiality, integrity or other security properties on the exchanged assertions. SAML assertions can be exchanged as security tokens within WSSE compliant SOAP messages.

XACML is the XML specification for expressing fine grained access policies in XML documents [10]. XACML is used to compose the rules which an access control mechanism has to examine in order to decide on the permission for the attached content and a set of subjects. An access control list in XACML is composed of a *subject* (e.g., user, group, ...), a *target object*, e.g. a document or an element within a XML document, a permitted action (e.g. read, write, ...), and a *provision*, expressing the action executed after the activation of a rule.

In the same spirit, a fine grained model for expressing authorization information during a SOAP request has been presented in [4]. XML credentials are enclosed in the header of SOAP messages, which are processed by the access control filter. The filter intercepts the requests and evaluates them w.r.t. the specified information and the information contained in a repository.

3 The Framework

Briefly we recall the metering protocol discussed in [1]. A *metering system* consists of n clients, say $\mathcal{C}_1, \ldots, \mathcal{C}_n$, interacting with an audit agency $\mathcal{A}$ and a server $\mathcal{S}$. The audit agency wants to hold a measure of the number of times the clients accesses a service provided by the server S. The players agree on a one-way hash function $\mathcal{H}$ with the *pre-image resistance*, *2nd-pre-image resistance collision resistance* properties [7]. The metering scheme we present here is based on the authentication of clients. Indeed clients which are previously registered by the audit agency can access the services provided by the server S.

The operations of the basic system are structured in the following way:

Initialization. The Initialization phase starts when a client C contacts the audit agency and requests access to the provided service. A number of subsequent operations are started by each of the players:

- The Audit Agency calculates a random seed, say w_0, and computes the the value of the *kth* application of the hashing function $\mathcal{H}$ on w_0, that is $w_k = \mathcal{H}^k(w_0) = \mathcal{H}(\mathcal{H}^{k-1}(w_0))$, where $\mathcal{H}^2(w_0) = \mathcal{H}(\mathcal{H}(w_0))$, and stores the tuple $[id_C, k, w_0]$ holding the seed and the number of guaranteed accesses with the client identifier id_C. Then it sends the two tuples $[id_C, k, w_0]$ and $[id_C, w_k]$ to the client C and the server S, respectively.
- The Client stores the tuple sent by A and retrieves the initial seed w_0, the number k of accesses it is registered for, and generates and stores the k values $w_1 = \mathcal{H}(w_0), w_2 = \mathcal{H}(\mathcal{H}(w_0)), \ldots, w_k = \mathcal{H}(w_{k-1})$
- The Server stores the tuple received from A in its database of registered clients, associating it with a counter L_C initially set to 0;

Interaction. Interaction happens when the client C visits the server site S and wants to access the restricted services offered by S,

- The client C, sends the token $wk - j = \mathcal{H}^{k-j}(w_0)$ for the *jth* access, and updates the access counter decrementing its value;

- The server S on the reception of the token, performs access control, verifying that the value resulting form the application of the hash function $\mathcal{H}$ to the received token matches the last stored value for the client C, that is $\mathcal{H}(w_{k-j}) = w_{k-j+1}$; if the result is positive then he updates its store with the new received token w_{k-j} and increments the visitor's counter L_C;

Verification. The verification phase begins when the server S claims the payment after a certain number of visits received in a certain period of time previously agreed with the agency.

- For client C the server sends to A a tuple (id_C, W, L_C), where id_C is the client's identifier associated with the last stored authentication token W and L_C is client's counter;
- The agency A verifies that the value returned from S for C, say W equals to $\mathcal{H}^{k-L_C}(w_0)$.

4 The Architecture

In this section we describe an architecture to implement the framework enabling an authenticated user to access and use restricted Web services. As depicted in Figure 1, our system consists of three components which expose the different services needed to realize all the phases of the protocol:

- **Client:** The Client interacts with the agency in order to receive a valid authentication token, granting him the number of accesses he can perform to a given service provided by the Server. The client produces SOAP requests and receives SOAP responses.
- **Audit Server:** The Audit Server exposes two services: one is called by a Client to get a a valid authentication token; the second service is invoked by a Server in order to get paid for the number of serviced requests to a given client.
- **Server:** The Server exposes a number of restricted and unrestricted services which are called in SOAP requests. For restricted services, access control is performed by analyzing the SOAP request. Additionally, he exposes a synchronization service, which can be called by a client in order to get realigned with the last token stored by the Server.

Both Server and Audit Server are connected to databases where the data needed to associate a client with its token and the number of bought accesses are stored.

In the following we describe the different kinds of interaction occurring during the different phases of the protocol.

Client/Audit Server. To invoke a restricted Web service, a client has to contact the audit server in order to register and get a valid authentication token. According to the business model, the token is issued after that some duty is fulfilled by the client, e.g., a payment is made. To get the token the client application

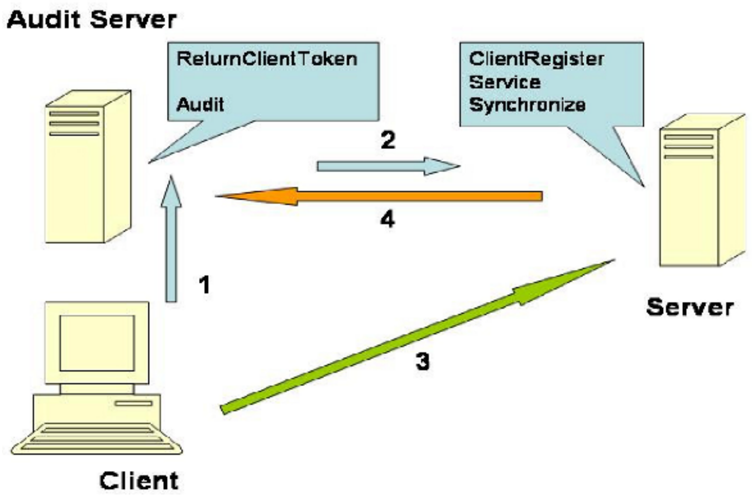


Fig. 1. The Actors and the Services provided in the Framework for Authenticated Web services. (1) and (2) denote the interaction during the Initialization phase. (3) denotes the Client/Server interaction. (4) denotes the Server/Audit Server interaction

issues a SOAP request to the *ReturnClientToken* service on the Audit Server. The SOAP response contains a *ClientId* field which uniquely identifies the client, a *NumAcc* field holding the number of allowed accesses, and a *Seed* field which is randomly computed. All those information are stored by the Audit Server and the Client in the Data Repository and Local Repository, respectively.

Audit Server/Server. After a successful client registration, the Audit Server contacts the Server in order to transmit the data associated with the freshly registered client. To this purpose the *ClientRegister* service is invoked. The SOAP request contains the client's identifier and the last value of the hash chain associated with the client, accordingly to the protocol described in the previous section. Such values are retrieved by the Server and stored in the Server Repository.

Client/Server. Each time the client application invokes a restricted service, the SOAP message containing the request has to be manipulated in order to provide authentication information in a transparent way for the user. The component responsible for such processing is the client side header *handler*, which intercepts the SOAP message, retrieves the authentication token from the Client Repository and constructs the header of the outgoing SOAP request, containing both the identification and the authentication tokens (Figure 2).

On the other side, the Server decides whether to give or not access to the restricted service on the basis of the authorization information contained in the SOAP request. To this purpose, the server side header handler retrieves the information contained in the header of the received SOAP message, getting both

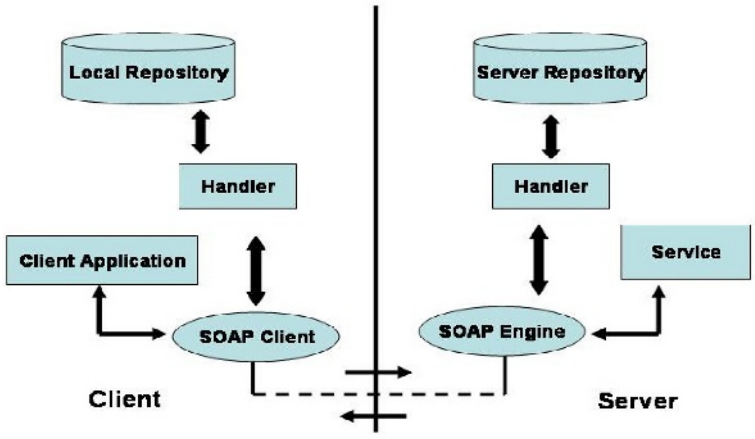


Fig. 2. The Client/Server Interaction

the identification and authorization tokens. The Server retrieves also the data stored in the Server repository and associated with the client identifier. If the value computed after the application of the hash function to the value retrieved from the SOAP header matches with the value retrieved from the database, the Server satisfies the service request, and passes the control to the execution of the service. In any other case, i.e., the header is empty, or the client identifier is not found into the Server repository, or the computed values do not match, a SOAP fault is raised, rejecting the unauthorized request.

Client and Server interact also if they get out of synchronization during the token exchange. The Server exposes a *Synchronization* service, which is invoked whenever a client request is rejected with a `<Wrong Value Hash>` message. In this case the Server is requested to provide the current token associated with a given client identifier, so that the Client can follow the chain of values till the returned value is encountered. The counter and the hash values are then realigned, such that a valid request can be successively constructed.

Server/Audit Server. Each time a Server wants to get paid for a given number of serviced requests, he has to contact the Audit Server and show the token received from the client as a proof. To this purpose, the Audit server exposes the *Audit* service. Such a service is invoked by the server which has to pass as parameters the client identifier, the authentication token held for that client, and the number of registered accesses. If the data match with the values held by the Audit Server in the Data Repository, the request is successfully served, and the corresponding action (a money transfer or a payment) is performed.

5 Implementation

Our prototype has been implemented using Java (version 1.4.2) as development language. As application server we used Tomcat (version 4.1), usually referred to as the official implementation for the Java Servlet and JavaServer Pages technologies. The SOAP engine is provided by Apache Axis (version 1.1) which offers a framework for constructing SOAP processors, a server which plugs into servlet engines (such as Tomcat), extensive support for the Web Service Description Language (WSDL), together with a set of other useful tools for the development phase.

The standard technique to modify SOAP messages is to act on a filter which intercepts every invocation addressed to the service provider. Such filter, which is usually referred as SOAP *Handler* has been used to enclose (retrieve, resp.) the authentication information within a SOAP request (response, resp) on the client side and retrieve (or enclose) the same information on the server side. For a Java implementation, this amounts to write a class implementing the `java.xml.rpc.handler.Handler` interface, and the code for the `handleRequest` method. As example, in Figure 5, the SOAP request for the restricted service *ComputeSum* with parameters 3 and 2 is showed; the authentication token for the Client associated to the id 239 is contained within the `<hash>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Header>
      <ns0:authHeader1
         xmlns:ns0="http://localhost:8080/axis/services/Authenticate">
         <ns0:id>239</ns0:id>
         <ns0:hash>5sPdYwQo/VSDQXK4/Sc1/tlBbaQ=</ns0:hash>
      </ns0:authHeader1>
   </soapenv:Header>
   <soapenv:Body>
      <ns1:computeSum
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="urn:Sum">
         <in0 xsi:type="xsd:int">3</in0>
         <in1 xsi:type="xsd:int">2</in1>
      </ns1:computeSum>
   </soapenv:Body>
</soapenv:Envelope>
```

Fig. 3. The SOAP message including the authentication token for the ComputeSum Service

The repositories have been implemented as databases accessed through the JDBC API. In the prototype implementation proposed, the data held by the Server, the Audit Server and the Client have been stored in separate databases holding one table for each subject. In a real implementation, the data related to the client could be simply stored in a file accessed through an enabled applet (obeying to a modified local security policy). A preliminary version of the implemented services has been published through the Universal Description, Discovery, and Integration (UDDI) Service, which is the industry-wide effort to bring a common standard for business-to-business(B2B) integration [11]. The purpose of UDDI is to allow users to discover available web services and interact with them dynamically.

6 Discussion

The specification released in January 2004 by the Oasis technical committee introduces a set of standard SOAP extensions which should enable applications to exchange messages securely [12]. The document specifies three basic mechanisms to ensure message confidentiality, message integrity and the ability to attach security tokens to standard messages. As reported in the document, these mechanisms do not provide a self-contained security solution for Web services, but are the building blocks which can be used together with other Web service extensions or standard techniques to provide security guarantees for applications.

Instead of customizing the SOAP header to implement the metering protocol, it is possible to use the standard extensions defined in the above mentioned specification. The security related information are included in a `<wsse:Security>` header block. To enclose security tokens, the SOAP Message Security specification defines two types of security tokens:

- *username*, which can be used to specify in a standard way a username together with some optional information;
- *binary* security tokens, which can be used to convey the data composing the token itself (e.g. aXS.509 certificates or Kerberos tickets [14,15]).

The information needed to execute the metering protocol can be enclosed then within the two elements `wsse:UserName` and `wsse:BinarySecurityToken` as reported in Fig. 6. Note that the attribute `EncodingType` of `wsse:BinarySecurityToken` defines the encoding format of the binary data.

7 Conclusions

Web services are standardizing the way applications communicate through the World Wide Web. The expectation is that complex business problems will be solved by the interconnection of competing Web service-enabled systems performing intra and inter-enterprise computing [3].

As soon as Web services become available, the problem of counting and granting accesses to restricted resources becomes more and more challenging. In this

```
<S11:Header>
   <wsse:Security xmlns:
      "http://schemas.xmlsoap.org/ws/2002/04/secext.xsd">
      <wsse:UsernameToken>
         <wsse:Username>Client Identifier<wsse:Username>
      </wsse:UsernameToken>
      <wsse:BinarySecurityToken
         EncodingType="wsse:Base64Binary">
            Authentication Token
      </wsse:BinarySecurityToken>
   </wsse:Security>
</S11:Header>
```

Fig. 4. The header of a SOAP request carrying both Username and Authentication Tokens

work we presented a framework allowing audit servers to measure accurately the number of requests that authenticated clients can perform on service providers. The implementation is straightforward and transparent for the actors of the systems, following the rules of services interaction.

The framework opens the door to the development of new business models, building on the application server provider model, where software and computing are offered on a subscription bases. We plan to extend our model to other situations, such as grid computing based Web Services, seen as the future platform for the provision of application functionality. In this scenario, Web Services are developed by the open source community who are remunerated by some form of electronic currency. The framework we propose could be exploited to implement such kinds of services, giving both to the subscripted users and to the service providers guarantees on the correctness of the behavior of the participants to the scheme.

References

1. C. Blundo and S. Cimato, A Software Infrastructure for Authenticated Metering *IEEE Computer*, April 2004.
2. G. Brose, A Gateway to Web Services Security – Securing SOAP with Proxies. In *Proceedings of International Conference on Web Services-Europe 2003 (ICWS-Europe 2003)*, Lecture Notes in Computer Science, vol. 2853, pp. 101-108, Springer, 2003.
3. M. Chen, A. N. K. Chen and B. Shao, The Implications and Impact ow Web Services to Electronic Commerce Research and Practices. *Journal of Electronic Commerce Research*, Vol. 4, N. 4, 2003.
4. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi and P. Samarati Securing SOAP E-Services. *International Journal of Information Security*, vol.
5. D. Geer, Taking Steps to Secure Web Services *IEEE Computer*, pp. 14-16, vol. , October 2003.

6. Leslie Lamport. Password authentication with insecure communication. In *Communications of the ACM*, volume 24, pages 770–771, 1981.
7. A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
8. Moni Naor and Benny Pinkas. Secure and efficient metering. In K. Nyberg, editor, *International Conference on the Theory an Application of Cryptographic Techniques (Eurocrypt '98)*, volume 1403 of *Lecture Notes in Computer Science*, pages 576–590, Espoo, Finland, 1998. Springer-Verlag, Berlin.
9. OASIS Committe Specification 01, W. Maler, P. Mishra and R. Philpott Editors, Assertions and Protocol for the OASIS Security Assertion MArkup Language (SAML) V. 1.1, September 2003
10. OASIS Standard. Extensible Access Control Markup Language, November 2002
11. OASIS Standard. UDDI version 2.04 API Specification, July 2002
12. OASIS Standard. Web Service Security: SOAP Message Security, Janauary 2004
13. OASIS Standard. Web Service Security: Username Token Profile 1.0, March 2004
14. OASIS Standard. Web Service Security: X.509 Certificate Token Profile, March 2004
15. OASIS Working Draft 10, "Web Services Security SAML Token Profile", April 2004
16. Ron Rivest and Adi Shamir. Payword and micromint: Two simple micropayment schemes. In *International Workshop on Security Protocols*, 1996.
17. S. Mysore, Securing Web Services – Concepts, Standards, Requirements. SUN Microsystems, October 2003
18. W3C Recommendation, XML Signature Syntax and Processing, February 2002.
19. W3C Working Draft, "XML Encryption Syntax and Processing,", March 2002

Development of Semantic Web Services at the Knowledge Level

Asunción Gómez-Pérez[1], Rafael González-Cabero[1], and Manuel Lama[2]

[1] Departamento de Inteligencia Artificial, Facultad de Informática.
Campus de Montegancedo s/n, Universidad Politécnica de Madrid.
28660 Boadilla del Monte, Madrid. Spain.
asun@fi.upm.es, rgonza@delicias.dia.fi.upm.es

[2] Departamento de Electrónica e Computación, Facultad de Física.
Campus Universitario Sur s/n, Universidade de Santiago de Compostela.
15782 Santiago de Compostela, A Coruña.
lama@dec.usc.es

Abstract. Web Services are interfaces to a collection of operations that are network-accessible through standardized XML messaging, and whose features are described using standard XML-based languages. Semantic Web Services (SWS) describe semantically the internal structure and the functional/non-functional capabilities of the services, facilitating the design and evaluation of SWSs based on that semantic description of the features of the services. To enable users to design and compose SWSs at the knowledge level, the ODE SWS framework has been proposed. That framework uses problem-solving methods to describe the functional and structural features of the SWSs. In this work, we present a description of the ODE SWS environment as an implementation of the ODE SWS framework. Specially, we focus on the description of the capabilities of the SWSDesigner, the tool of the ODE SWS environment that enables users to design graphically SWSs through different but complementary views of the services.

1 Introduction

Web Services (WSs) are interfaces that describe a collection of operations that are network-accessible through standardized Web protocols, and whose features are described using a standard XML-based language ([1,2]). These features are the following: (1) communication features describe the protocols required to invoke the service execution; (2) descriptive features detail the e-commerce properties; (3) functional features that specify the capabilities, enabling thus for an external invoking agent to determine whether the service execution can obtain the requested results; and (4) structural features describing the internal structure of a composite service, that is, which are its structural components and how those components are combined among them to execute the service.

In this context, the Semantic Web [3] has risen as a Web evolution where the information would be directly machine-readable to enable software agents to access to it. Following this approach, Web Services in the Semantic Web, so-called Semantic

Web Services (SWSs), must be described using an ontology that is expressed in a semantically enriched markup language [4]. This semantic description will facilitate external agents to understand both the functionality and the internal structure of the services to be able to discover, compose, and invoke SWSs [5]. The markup language could be OWL [6], but it must be combined with WS standard languages to be able to use the current infrastructure of the WS [7]. Following this approach, the OWL-S [8] specification (formerly DAML-S [9]) has been proposed to describe services in a semantic manner, using OWL in combination with WSDL [10] and SOAP [11].

However, as a previous step to the specification of SWS in a semantic Web-oriented language, the SWS should be designed at a knowledge or conceptual level [12] to avoid inconsistencies or errors among the services that constitute the SWS. In this context, SWS design consists in specifying the descriptive, functional, and structural features of a service. Currently, there are some proposals to edit/design SWSs, but the main drawback of these available editing tools is that they operate at the representation level, so the following problems may arise:

- These tools are language-dependent, like the WSMO Editor [13], this means that: (1) SWSs designed with these tools are less reusable, because the design can be constrained with the chosen language characteristics, meaning that the basic separation between design-implementation phases is broken; and (2) the designs are more prone to inconsistencies or errors that designs at the knowledge level.
- Many tools that claim to be SWSs editors are not actually more than mere ontology editors, like the widespread option of OWL-S development with the OWL plug-in for Protegé-2000 [14]. This option not only suffers from all the problems enumerated above, but also adds the problem of working with an ontology instantiation, not with a SWS-like structure.

To solve those drawbacks, we have proposed a framework, called ODE SWS [15], for design of SWSs at knowledge and independent-language level. This framework is based on: (1) a stack of ontologies that describe explicitly the different features of a SWS; (2) a set of axioms used to check the consistency and correctness of the ontology instances, and, thus, of the service represented by the ontologies; and (3) the assumption that a SWS is modeled as a problem-solving method (PSM) that describes how the service is decomposed into its components, and which is the control of the reasoning process to execute the service.

In this paper, we describe the architecture of ODE SWS for design of SWSs, and specially, we focus on the capabilities of SWSDesigner, a tool that is the user interface of ODE SWS. SWSDesigner uses PSM-alike views for describing the SWSs and, the service is designed thus by filling the SWS definition; specifying its associated task (interaction and logic diagrams); creating the overall task hierarchy (decomposition diagram); building both the associated method internal dataflow (knowledge flow diagram) and the coordination of the execution of its subtasks (control flow diagram).

This paper is structured as follows: in the following section the ODE SWS framework is presented; then, we show the functionalities of the environment that implements the ODE SWS framework: its architecture and current modules are described, paying special attention to the SWSDesigner graphical editor. Finally, we summarize the main contributions of the paper.

2 ODE SWS Framework

The aim of designing SWSs is making explicit the features previously mentioned and specially each one of its structural components, guaranteeing the correctness of the proposed design, and avoiding the inconsistencies. For that, we will need to perform inferences about the service features to determine whether the proposed design is correct. This means that the service features (and the service itself) should be explicitly and semantically described, and, for it, the use of ontologies seems to be the most appropriate solution. This approach has been also followed by other authors [8], who use a semantic-enriched markup language to create an ontology (OWL-S) that describes the service features.

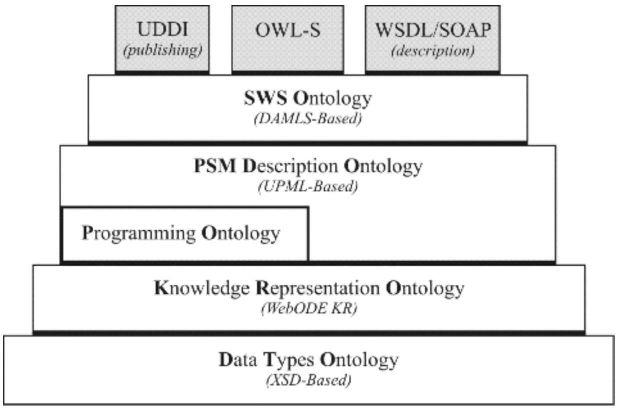


Fig. 1. Ontology set identified in the ODE SWS framework [15] to SWS design. These ontologies have been developed based on well-known specifications and *de facto* standards

Figure 1 shows the stack of ontologies proposed in the ODE SWS framework [15,23] to describe all the features of a SWS (and the service itself). To construct these ontologies well-known specifications or *de facto* standards have been used. This will favor the interoperability of the ODE SWS framework with applications or solutions constructed following one of those specifications.

Problem-Solving Method Ontology

Problem-Solving Methods (PSM) [16,17] are knowledge components reusable among different, but related, domains and tasks. The Unified Problem-solving Method Language (UPML) [18] is a *de facto* standard that describes the components of a PSM as: (1) *tasks*, they describe the operation to be solved in the execution of a method that solves such task, specifying the input/output parameters and the pre/post-conditions (competence) required to be applicable (this description is independent of the method used for solving the task); (2) *methods*, elements that detail the control of the reasoning process to achieve a task; and (3) *adapters* [19] specify mappings among the knowledge components of a PSM. The adapters are used to achieve the reusability at the knowledge level, since they bridge the gap between the general description of a PSM and the particular domain where it is applied.

Based on the UPML specification we have created a PSM ontology that enhances the description of the UPML elements. Some additions or differences are: new relationships between tasks and methods are defined, a set of program elements to specify the control flow of a composite method are defined, we define a combination of them that allow us to derive several basic workflow-like patterns [20,21], like sequence or exclusive and multiple choice, the domain ontology is managed in a different manner.

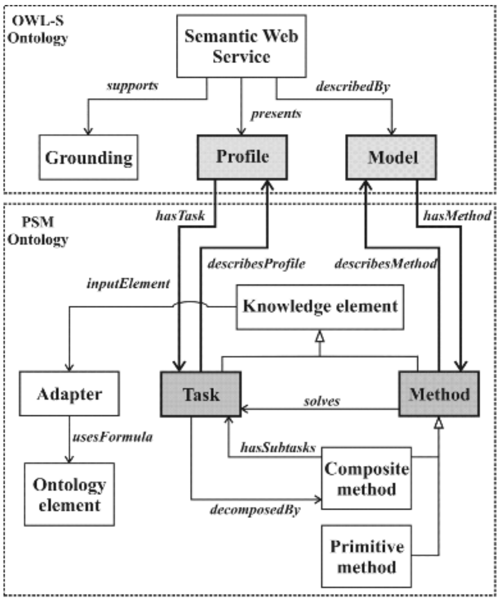


Fig. 2. Semantic Web Service ontology and its relationship with the PSM description ontology, where tasks will be used to represent the functional features of a SWS, and methods to describe the internal structure and control flow of that service [15]

Semantic Web Service Ontology

As Figure 2 shows, SWS ontology replicates the upper-level concepts of the OWL-S ontology, so we consider: (1) *profile* contains all the descriptive and functional features, and it is establishes a relationship with a task of the PSM ontology; (2) *model* deals with the internal structure of the service, which will be executed by a method that defines an operational description to solve the task related to the functional features of the service; and (3) *grounding*, which specifies the access protocol and the necessary message exchanges to invoke the service.

2.1 Framework for SWS Design and Composition

The proposed framework for SWS [15] design and composition is directly based on the stack of ontologies that describe the features of a SWS. The ODE SWS framework details how to create a SWS with the capabilities required by an external agent or user. The main elements of the framework are (Figure 3):

1. *Instance model.* Design of SWSs means to instantiate all the ontologies that describe what a service is: the domain ontology used by the service is instantiated in both data types and knowledge representation ontology, whereas the service features are instances of both PSM and SWS ontologies. The whole instances constitute a model that specifies the SWS at the knowledge level.
2. *Checking model.* Once the instance model has been created, it is necessary to guarantee that the ontology instances do not present inconsistencies among them. Design rules will be needed to check this, particularly when ontology instances have been created automatically.
3. *Translate model.* Although a service is modeled at the knowledge level, it must be specified in a SWS-oriented language to enable programs and external agents to access to its capabilities.

This framework enables the (semi) automatic composition of SWSs using (1) PSM refiners and bridges to adapt the PSM ontology instances to the required capabilities of the new service; and (2) design rules to reject both PSM and SWS ontology instances that present errors or inconsistencies among them. Design rules are used to reduce the service candidates combined to obtain the new service.

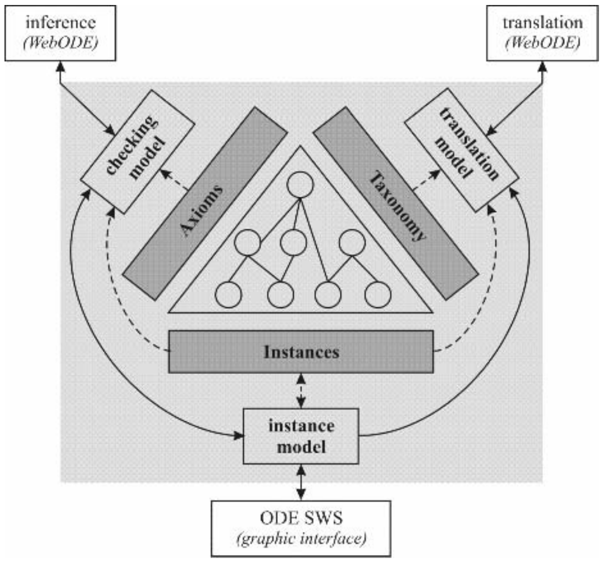


Fig. 3. Framework for design and composition of SWSs based on the ontologies that describe the service features

3 ODE SWS Environment

Following the ODE SWS framework, a highly modularized, scalable and dynamic environment for development of SWSs called ODE SWS [22,23] has been implemented. Its architecture and functionalities will be thoroughly explained in the forthcoming sections.

3.1 ODE SWS Architecture

The architecture of the ODE SWS, as Figure 4 shows, is composed of three main layers, which reflect the layers introduced in a software design [24]:

Presentation Layer. This layer is about how to handle the interaction between the user and the software system. This layer is entirely composed by the SWSDesigner graphical editor that manages a graphical model (SWSGM) of the SWS. Its main functionalities are: (1) the appropriate management and representation of the model; (2) graphical processing of all the possible interactions among the elements that compose such model; and (3) the management of the graphical elements of the ODE SWS environment. Any other functionality is delegated to the appropriate ODE SWS modules of the domain layer.

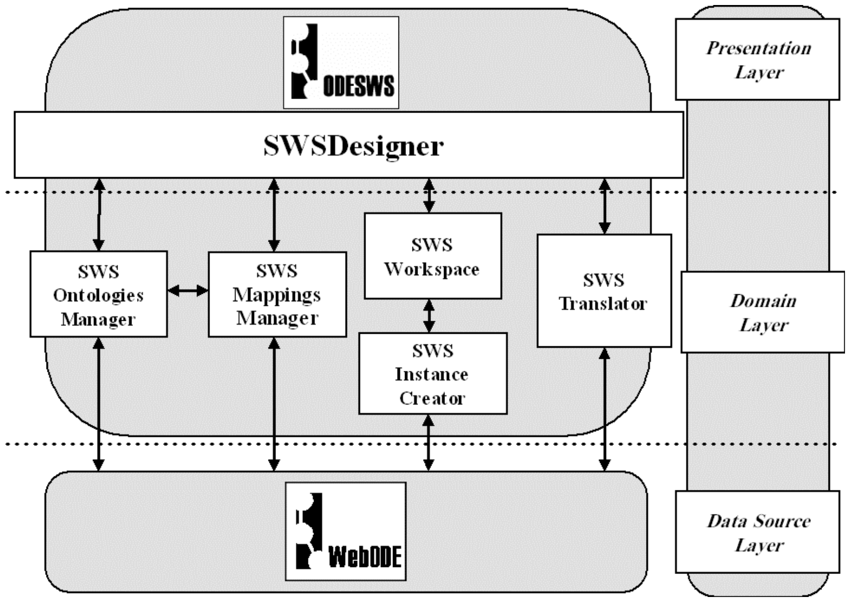


Fig. 4. Software architecture of the ODE SWS environment for the development of Semantic Web Services

Domain Layer. This layer contains all the components that work in the domain of the ODE SWS application (i.e., SWSs), and, consequently, most of ODE SWS modules will be located in this layer. SWSDesigner directly will invoke the execution of the ODE SWS modules to support the execution of an operation needed to guarantee a correct design of the service. The intended functionalities of each of these components are:

- *SWSOntologiesManager.* The purpose of this module is both to offer a uniform manner for accessing to ontologies implemented in different languages, and to enable ODE SWS to access to different repositories of ontologies. Therefore, this module guarantees the language and technology independence for the ontologies managed in the development of the service. Currently this module can manage either ontologies implemented in RDF(S), DAML+OIL and OWL, or ontologies stored in the WebODE platform [25]. Note that this module is part of the domain layer because the ontologies that it manages are stored in the source repository.
- *SWSMappingsManager.* The objective of this module is to manage mappings, which will be (semi) automatically defined between the elements of the task ontologies and the elements of the method and domain ontologies.
- *SWSWorkspace.* While the user is developing a SWS, an incomplete (even inconsistent) service may be stored and managed. This module performs these activities, enabling thus the store and recovery of ongoing SWSs.
- *SWSInstanceCreator.* This module creates the instances of the stack of ontologies that describe the SWSs from the graphical representation of the service generated by the SWSDesigner.
- *SWSTranslator.* This module implements the translation model of the framework. Once the SWS has been modeled at the knowledge level, it must be translated to a SWS-oriented language to enable other programs or agents to understand its capabilities. So, once the SWS has been created using SWSDesigner, the user asks for a list of available translators. After that, this module receives which is the selected translator, the desired output format, and the graphical representation model of the SWS to be translated. SWSTranslator generates the translation and, additionally, it incorporates information about the different problems that may have arisen during the translation process. Note that it may use its own inner translators or even external translation services, as it does with the export services of the WebODE platform. At present, the translation to OWL-S and WSDL are available.

Data Source Layer. In this layer, other applications act on behalf of the ODE SWS environment to provide support for operations do not implemented in the environment. For the ODE SWS environment, this layer will be just composed by the WebODE platform, which will provide services for the management and access to the ontologies used in the development of the SWSs.

3.2 SWSDesigner

SWSDesigner is a graphical editor based on the assumption that the design and development of a service should be performed from different, but complementary, points of view. In the ODE SWS framework, as we have already stated, a service is described instantiating a set of ontologies that describe all the SWS features. Taking this into account, SWSDesigner provides a user-friendly graphical interface with which the user is completely unaware of the instantiation of the SWS ontologies. In this way, we achieve the following objectives: (1) the service is defined at a high level of abstraction using PSMs to model the service features. This modelling enhances the quality of the design, eases its evaluation and validation processes, and favours its reuse; and (2) this design process is far more simple and less error prone than manipulating directly instances of the different description ontologies.

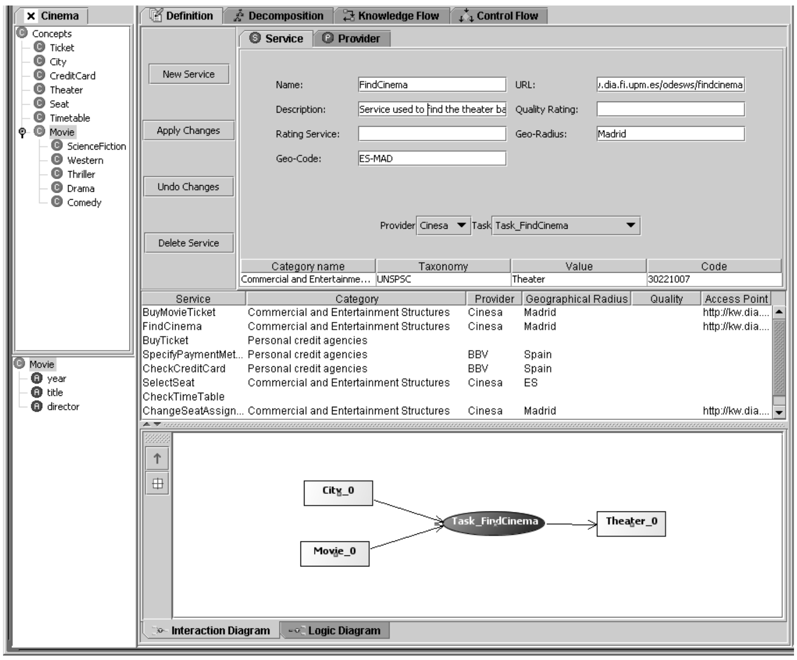
As stated in the ODE SWS framework section, a service has its own descriptive (non-functional) properties that are described by a task, whereas the description of how this task (and the service) is supposed to be carried out is specified by a method. As we will see, all this information can be gathered from the different views that the SWSDesigner manages. To illustrate the functionalities of this tool, in the following sections we will define a service for selling movie tickets.

Service Definition Panel. The service definition panel includes several kinds of information related to the declarative (or non-functional) service features and the properties of the providers. It includes (see Figure 5):

– *Service definitions*. The information of the service contained in its definition will include: (1) information needed for the identification and description of the service (i.e., name, description and URL); (2) descriptive features that will detail its e-commerce properties (i.e., geographical location, commerce classification, service provider, etc.); and (3) a summary table of services that contains information about all the services in a condensed view.
– *Provider definitions*. The provider forms include information such as: (1) information necessary for the identification and description of the provider (i.e. name, description); (2) descriptive features that detail its e-commerce properties (its geographical radius and code, and its commerce classification); (3) the contact persons that can be assigned (and the ones that have been already assigned) to the providers, including all the contact data (name, phone, e-mail, fax, etc.); and (4) summary table of providers that contains information about all the providers in a compacted manner.

Furthermore, in the service definition panel, the description of the functionality (input/output data) and the competence of the task associated with the service is introduced. This description is achieved with two diagrams:

– *Interaction diagram*. In this diagram the input and output roles of the task are defined. A role is an ontology element (concept or attribute), and it can be easily inserted into the diagram by just dragging-and-dropping them from the ontology tree. Once it has been deployed, an edge between the role and the task can be created; depending on the direction of this connection, the role is an input or an output of the task. Unassigned roles are allowed.
– *Logic diagram*. In this diagram, the pre/post-conditions (competence) required by a task to be applicable, and the effects of the execution of such task in the state of the world are set. All these logical conditions are represented as different kinds of cells of the graph, with edges going from the condition toward the task in the case of pre-conditions, and with connections from the task to the cells when effects and post-conditions are introduced.



Name:	FindCinema	URL:	›.dia.fi.upm.es/odesws/findcinema
Description:	Service used to find the theater ba	Quality Rating:	
Rating Service:		Geo-Radius:	Madrid
Geo-Code:	ES-MAD		

Provider Cinesa Task Task_FindCinema

Category name	Taxonomy	Value	Code
Commercial and Entertainme...	UNSPSC	Theater	30221007

Service	Category	Provider	Geographical Radius	Quality	Access Point
BuyMovieTicket	Commercial and Entertainment Structures	Cinesa	Madrid		http://kw.dia....
FindCinema	Commercial and Entertainment Structures	Cinesa	Madrid		http://kw.dia....
BuyTicket	Personal credit agencies				
SpecifyPaymentMet...	Personal credit agencies	BBV	Spain		
CheckCreditCard	Personal credit agencies	BBV	Spain		
SelectSeat	Commercial and Entertainment Structures	Cinesa	ES		
CheckTimeTable					
ChangeSeatAssign...	Commercial and Entertainment Structures	Cinesa	Madrid		http://kw.dia....

Fig. 5. Service definition panel with the descriptive and functional features of *FindCinema*

In the Figure 5, the service definition panel of the SWS *FindCinema* is shown. All the descriptive information of the service can be introduced here. The task information is introduced in the interaction diagram, indicating that the *Task_FindCinema* task receives two input roles: *Movie_0*, which is an instance of the concept *Movie*, and *City_0*, which is an instance of the concept *City*. The output is an instance of the concept *Theater*. These elements have been dragged-and-dropped from the ontology trees; one of them shows the concepts, and the other shows the attributes of each concept. Note that this ontology could be other ontology different of the domain ontology. As it will be explained, mappings between domain ontology elements and the inputs/outputs of the task are set in the knowledge flow diagram.

Decomposition Diagram. The decomposition diagram enables the user to specify the task hierarchy, that is, how the defined tasks are related to each other. This diagram will be synchronized with the task tree, which will be visible for every SWSDesigner view in the top right corner of the screen. There will be two possible representation options:

– *A task can be decomposed by a method in one or more subtasks*. This view allows user to specify the subtasks in which a task is decomposed by a composite method that solves such task. This kind of relations is represented as an edge from the subtasks to the task.

– *A task specialises another task*. In this case, the user will have to introduce an adapter, more precisely a task-refiner, among the implied tasks. The properties of the task-refiner could be edited by the user after its inclusion in the diagram.

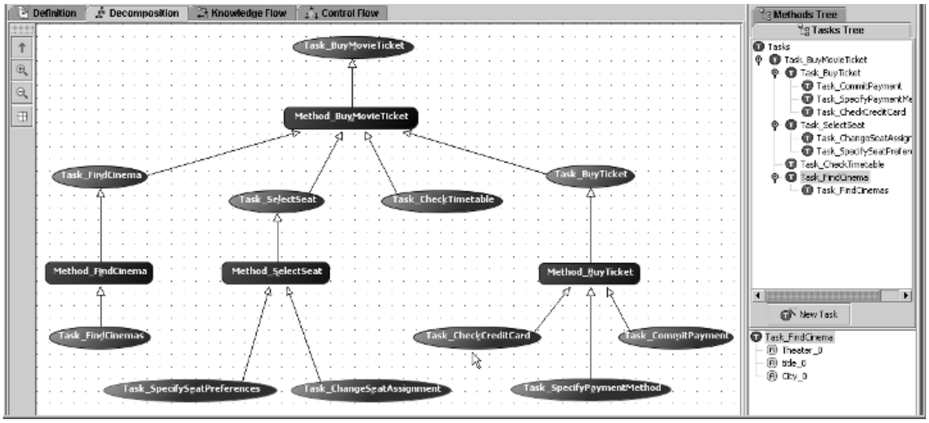


Fig. 6. Decomposition diagram for the *Task_BuyMovieTicket* task that describes the functional features of a given SWS

Figure 6 shows the overall task hierarchy for the selling movie ticket example. Thus, to solve the composite task *Task_BuyTicket*, the tasks *Task_CheckCreditCard*, *Task_CommitPayment*, and *Task_SpecifyPaymentMethod* might need to be accomplished. In the right corner of the screen, the task tree is shown. It allows the access to the tasks in the other views. Note that this view does not impose any order of execution at all; it is set in the control flow diagram.

Knowledge Flow Diagram. The knowledge flow diagram shows the data flow among the tasks in which a given composite method has been decomposed. It relates (1) the inputs/outputs of a method to the inputs/outputs of its various component subtasks; (2) the inputs/outputs of a subtasks of a given method with the inputs/outputs of the other subtasks of such method; and (3) the domain ontology to the method and tasks. Therefore, there will be three basic elements: roles, tasks and edges. Tasks can be placed easily, just dragging-and-dropping them into the graph from the tasks tree. The management of roles and edges must be made more carefully. Roles will wrap an ontology element, and can be inserted into the diagram by just dragging-and-dropping them from the ontology tree. Depending on different possible combinations, three plausible scenarios are distinguished:

– *Unassigned Roles*. When a role is isolated, that is, there are no connections either going out from it or pointing at it, the role does not affect the model at all. This kind of roles is allowed just for user convenience.
– *Inner Roles*. This situation occurs when a role is input of a task and at least output of another task. It might be viewed as an inner message between elements inside the method. Here, a domain-task bridge, which maps the ontology element wrapped by the role with the input/output of the affected tasks, is created.
– *External Roles*. These roles are either inputs or outputs of a task, but not inputs of a task and outputs of another tasks. Therefore, these roles must be provided to the method, if it is going to be invoked, because it is not a data flow between tasks, but these roles are related to output and output messages from or towards the method.

Two adapters are created in this situation thus, a domain-task bridge (as in the case of the inner role) and a domain-method bridge.

To ease the management of the different kind of roles and the different mappings that may appear, the knowledge flow diagram offers two useful tools:

– *Summary diagram*. This diagram is automatically updated each time a user does some kind of manipulation in the knowledge flow diagram (a role is inserted, an edge is removed, etc.). It shows just the external roles, allowing the special and separate handling of them.
– *Mapping generator*. When a user creates an edge from a role to a task (or *vice versa*), an adapter between this role and a role of the task interaction diagram is created. The role of the interaction diagram origin or destination of this binary relation must be directly asked via interface to the user. The mapping generator will obtain automatically the plausible candidates.

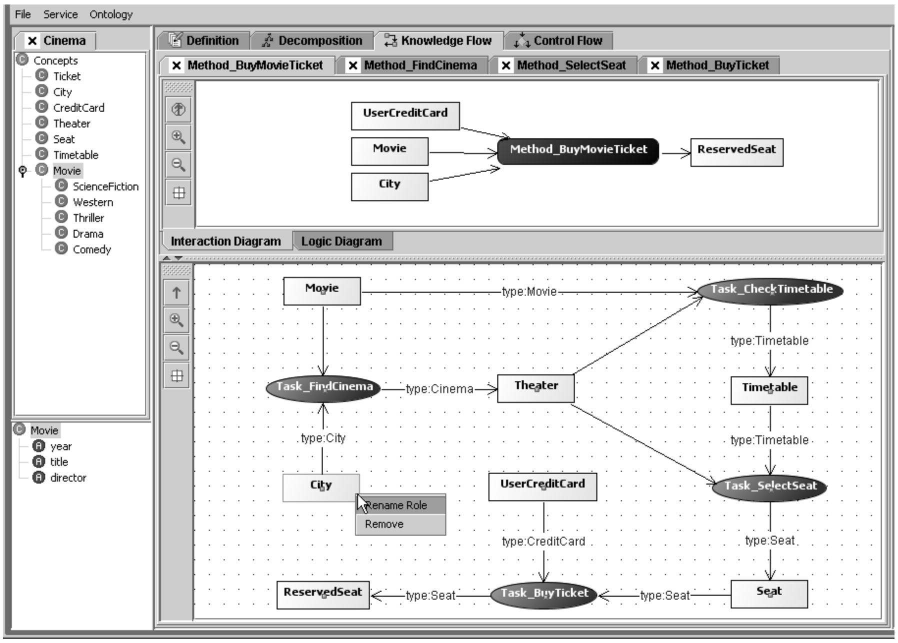


Fig. 7. Knowledge flow diagram for the composite method *Method_BuyMovieTicket* that shows the input/output interaction among the sub-tasks of the method

The knowledge flow diagram of the method *Method_BuyMovieTicket* is shown in Figure 7. The roles of this diagram are linked to the method (and domain) ontology elements, and each connection with a task has a mapping, a domain-task bridge between a domain element and a task input or output role. For example, in Figure 5 we have shown the interaction diagram of the task *Task_FindCinema*. Its output was *Theater_0*, an instance of the concept *Theater*. In the knowledge flow of the method *Method_BuyMovieTicket*, let us introduce an edge between the role *Theater* and the task *Task_FindCinema*. In that case, as Figure 8 shows, the Mappings generator will offer the candidate list (the output role *Theater_0*) and the user could create a new interaction diagram role to map with the role *Theater*. We choose *Theater_0*, and a domain-task bridge is created. It will map the *Theater* concept of the domain ontology with the *Theater* concept of the ontology that was used to create the task definition.

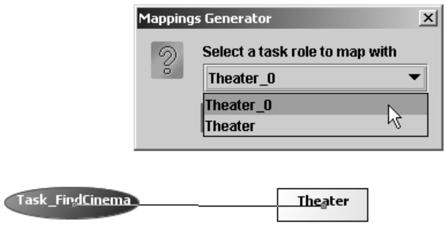


Fig. 8. The Mapping generator that presents the candidate list of possible mappings to be selected by the user

Control Flow Diagram. The control flow diagram is part of the method definition that solves the task related to the service. In this view, the user specifies the flow control of a method, where its sub-tasks are combined with programming structures to obtain a description of the service execution. In other words, this diagram describes how a composite method uses its sub-tasks in order to achieve its capability. The set of program elements that can be introduced in the control flow diagram are: (1) *condition*, depending on a logical condition, the true or false branch is chosen; (2) *conditional loop*, the true branch is executed till a determined boolean expression is false; (3) *split*, a new thread of execution is created for each output branch; and (4) *join*, synchronize or serialize various input execution tasks to one same output.

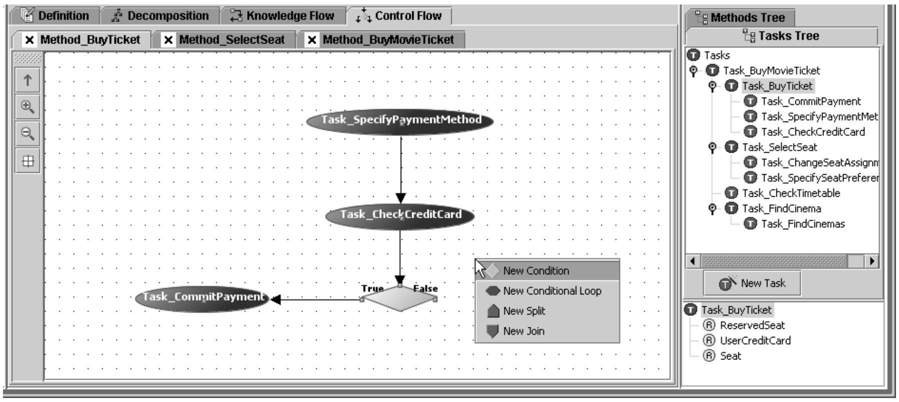
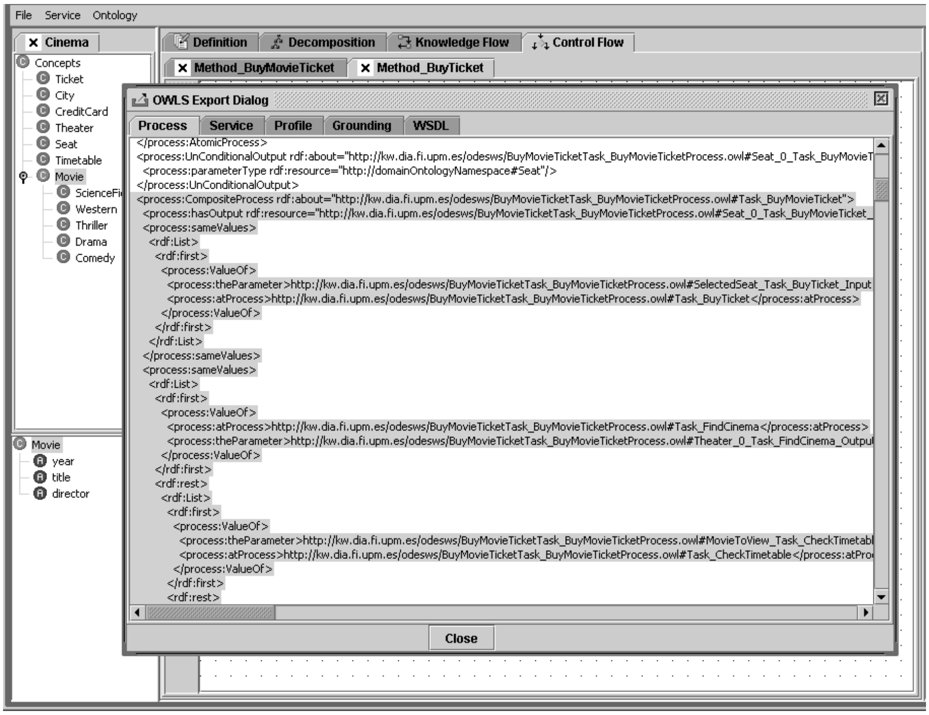


Fig. 9. Control flow diagram for the method that will solve the *Task_BuyTicket* task associated with a given service

Figure 9 shows the control flow diagram of the method *Method_BuyTicket*. First, the details of the payment are set. The next step is to check whether the credit card can afford this payment or not. If it is possible, this credit card must be charged.

Once the user has designed the service following all the complementary views provided by the SWSDesigner, it is necessary to translate that service from the graphical representation into a semantic-oriented language such as OWL-S or WSMO. To enable this translation, the SWSDesigner invokes the SWSInstanceCreator execution, which uses the graphical model of the SWS to create the instances of the SWS description ontologies. Then, the SWSTranslator is invoked to translate these instances into the language selected by the user (currently OWL-S). Figure 10 shows the OWL-S specification of the service *BuyMovieTicket* that has been automatically generated by the SWSTranslator.



```
</process:AtomicProcess>
<process:UnConditionalOutput rdf:about="http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Seat_0_Task_BuyMovieT
 <process:parameterType rdf:resource="http://domainOntologyNamespace#Seat"/>
</process:UnConditionalOutput>
<process:CompositeProcess rdf:about="http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Task_BuyMovieTicket">
 <process:hasOutput rdf:resource="http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Seat_0_Task_BuyMovieTicket_
 <process:sameValues>
  <rdf:List>
   <rdf:first>
    <process:ValueOf>
     <process:theParameter>http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#SelectedSeat_Task_BuyTicket_Input
     <process:atProcess>http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Task_BuyTicket</process:atProcess>
    </process:ValueOf>
   </rdf:first>
  </rdf:List>
 </process:sameValues>
 <process:sameValues>
  <rdf:List>
   <rdf:first>
    <process:ValueOf>
     <process:atProcess>http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Task_FindCinema</process:atProcess>
     <process:theParameter>http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Theater_0_Task_FindCinema_Outpu
    </process:ValueOf>
   </rdf:first>
   <rdf:rest>
    <rdf:List>
     <rdf:first>
      <process:ValueOf>
       <process:theParameter>http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#MovieToView_Task_CheckTimetab
       <process:atProcess>http://kw.dia.fi.upm.es/odesws/BuyMovieTicketTask_BuyMovieTicketProcess.owl#Task_CheckTimetable</process:atPro
      </process:ValueOf>
     </rdf:first>
     <rdf:rest>
```

Fig. 10. OWL-S specification of the service *BuyMovieTicket*. The SWSTranslator automatically generates this specification through the SWSDesigner

4 Conclusions

Current tools that enable users to design SWSs depend on the capabilities of representation and reasoning of a specific SWS-oriented language. Those tools are constrained by the language expressiveness; the service must be designed using the capabilities provided by the language in which will be expressed. Furthermore, users usually introduce both errors and inconsistencies, which could be minimized using tools that operate at the knowledge level.

Consequently, to solve these problems, in this paper we claim to design SWSs at the knowledge level, and to provide tools to facilitate the design SWSs in a language-independent manner. For it, we have developed the ODE SWS conceptual framework and the environment that supports such framework. Using the graphical interface of the ODE SWS environment, called SWSDesigner, users can introduce in an easy manner the descriptive and functional features of the service, as well as, the internal components and how those components are coordinated among themselves to execute the SWS. Once the service is completely designed, and following the ODE SWS framework, it will be checked to detect inconsistencies and/or errors that could be present in the user design. If they are not detected, user will select the language (currently WSDL and OWL-S are supported) in which the SWS will be expressed.

Acknowledgements. Authors would like to thank the Esperonto project (IST-2001-34373) for their financial support in carrying out this work.

References

1. Kreger, H.: Web Services Conceptual Architecture (WSCA 1.0). IBM Software Group. http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf (2001)
2. Curbera, F., Nagy, W.A., Weerawana, S.: Web Service: Why and How?. Proceedings of the OOPSLA-2001 Workshop on Object-Oriented Services. Tampa, Florida (2001)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American. 284 (2001) 34–43
4. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems. 16 (2001) 46–53
5. Hendler, J.: Agents and the Semantic Web. IEEE Intelligent Systems. 16 (2001) 30–37
6. Dean, M., Schreiber, G. (eds.): OWL Web Ontology Language Reference. W3C Candidate Recommendation. http://www.w3c.org/TR/owl-ref (2004)
7. Sollazo, T., Handshuch, S., Staab, S., and Frank, M.: Semantic Web Service Architecture – Evolving Web Service Standards toward the Semantic Web. Proceedings of the 15th International FLAIRS Conference. Pensacola, Florida (2002)
8. The OWL Services Coalition: OWL-S 1.0 Release: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.0/owl-s.pdf (2004)
9. Ankolenkar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T., Sycara, K., Zeng, H.: DAML-S: Web Service Description for the Semantic Web. Proceedings of the First International Semantic Web Conference. Sardinia, Italy (2002) 348–363
10. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Service Description Language (WSDL) 1.1. http://www.w3c.org/TR/2001/NOTE-wsdl-20010315 (2001)
11. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) Version 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508 (2000)
12. Newell, A.: The Knowledge Level. Artificial Intelligence. 18 (1982) 87–127

13. Lausen, H., Felderer, M., Roman, D. (eds.): Web Service Modeling Ontology (WSMO) Editor. http://www.wsmo.org/2004/d9/v01 (2004)
14. Noy, N., Fergerson, R.W., and Musen, M.A.: The knowledge model of Protégé-2000: Combining interoperability and flexibility. Proceedings of the 12th International Conference in Knowledge Engineering and Knowledge Management (EKAW'00). Lecture Notes in Artificial Intelligence, Vol. 1937. Juan Les Pins, Francia (2000) 17–32
15. Gómez-Pérez, A., González-Cabero, R., Lama, M.: A Framework for Design and Composition of Semantic Web Services. Proceedings of the AAAI Spring Symposium on Semantic Web Services. AAAI Press, Stanford, California (2004) 113–121
16. Benjamins, V.R., Fensel, D.: Special Issue on Problem-Solving Methods. International Journal of Human-Computer Studies. 49 (1998) 305–313
17. Motta, E.: Reusable Components for Knowledge Mdelling. IOS Press, Amsterdam, The Netherlands (1999)
18. Fensel, D., Motta, E., van Harmelen, F., Benjamins, V.R., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., Musen, M.A., Plaza, E., Schreiber, G., Studer, R., Wielinga, B.: The Unified Problem-Solving Method Development Language UPML. Knowledge and Information System. 5 (2003) 81–131
19. Fensel, D.: The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. Proceedings of the 10th Knowledge, Modeling and Management Workshop. Lecture Notes in Computer Science, Vol. 1319. Springer-Verlag, Berlin Heidelberg (1997) 97–112
20. van der Aalst, W.P., van Hee, K.: Workflow magament – Models, Methods, and Systems. MIT Press. Cambridge, Massachusetts (2002)
21. van der Aalst, W.P., ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases. 14 (2003) 5–51
22. Corcho, O., Fernández-López, M., Gómez-Pérez, A., Lama, M.: An Environment for Development of Semantic Web Services. Proceedings of the IJCAI-2003 Workshop on Ontologies and Distributed Systems. http://CEUR-ORG.com/Vol-71. Acapulco, México (2003) 13–20
23. Corcho O., Fernández-López, M., Gómez-Pérez, A., Lama, M.: ODE SWS: A Semantic Web Service Development Environment. Proceedings of the VLDB-2003 Workshop on Semantic Web and Databases. Berlin, Germany (2003) 203–216
24. Fowler, M.: Patterns of Enterprise Application Architecture. Addison Wesley (2003)
25. Arpírez, J.C., Corcho, O., Fernández-López, M., Gómez-Pérez, A.: WebODE in a Nutshell. AI Magazine. 24 (2003) 37–48

A Directory for Web Service Integration Supporting Custom Query Pruning and Ranking

Walter Binder, Ion Constantinescu, and Boi Faltings

Artificial Intelligence Laboratory
Swiss Federal Institute of Technology Lausanne (EPFL)
CH–1015 Lausanne, Switzerland
`firstname.lastname@epfl.ch`

Abstract. In an open environment populated by large numbers of heterogeneous information services, integration is a major challenge. In such a setting, the efficient coupling between directory-based service discovery and service composition engines is crucial. In this paper we present a directory service that offers specific functionality in order to enable efficient web service integration. Results matching with a directory query are retrieved incrementally on demand, whenever the service composition engine needs new results. In order to optimize the interaction of the directory with different service composition algorithms, the directory supports custom pruning and ranking functions that are dynamically installed with the aid of mobile code. The pruning and ranking functions are written in Java, but the directory service imposes severe restrictions on the programming model in order to protect itself against malicious or erroneous code. With the aid of user-defined pruning and ranking functions, application-specific ordering heuristics can be directly installed into the directory. Due to its extensibility, the directory can be tailored to the needs of various service integration algorithms. This is crucial, as service composition still needs a lot of research and experimentation in order to develop industrial-strength algorithms. Experiments on randomly generated problems show that special pruning and ranking functions significantly reduce the number of query results that have to be transmitted to the client by up to 5 times.[1]

Keywords: Incremental service integration, query pruning, service directories, service discovery, service ranking

1 Introduction

Service composition[2] is an exciting area which has received a significant amount of interest in the last period. Initial approaches to web service composition [25] used a simple forward chaining technique which can result in the discovery of large numbers

[1] The work presented in this paper was partly carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Science Foundation as part of the project MAGIC (FNRS-68155), as well as by the Swiss National Funding Agency OFES as part of the European projects KnowledgeWeb (FP6-507482) and DIP (FP6-507483).

[2] In this paper we use the terms 'service composition' and 'service integration' interchangeably.

of services. There is a good body of work which tries to address the service composition problem by applying planning techniques based either on theorem proving (e.g., Golog [19,20], SWORD [22]) or on hierarchical task planning (e.g., SHOP-2 [31]). The advantage of this kind of approach is that complex constructs like loops (Golog) or processes (SHOP-2) can be handled. All these approaches assume that the relevant service descriptions are initially loaded into the reasoning engine and that no discovery is performed during composition.

Still the current and future state of affairs regarding web services will be quite different since due to the large number of services and to the loose coupling between service providers and consumers we expect that services will be indexed in directories. Consequently, planning algorithms will have to be adapted to a situation where operators are not known a priori, but have to be retrieved through queries to these directories. Recently, Lassila and Dixit [16] have addressed the problem of interleaving discovery and integration in more detail, but they have considered only simple workflows where services have one input and one output.

Our approach to automated service composition is based on matching input and output parameters of services using type information in order to constrain the ways how services may be composed [10]. Our composition algorithm allows for *partially matching* types and handles them by computing and introducing *switches* in the integration plan. Experimental results carried out in various domains show that using partial matches decreases the failure rate by up to *7 times* compared with an integration algorithm that supports only complete matches [10].

We have developed a directory service with specific features to ease service composition. Queries may not only search for complete matches, but may also retrieve *partially matching* directory entries [8]. As the number of (partially) matching entries may be large, the directory supports *incremental retrieval* of the results of a query. This is achieved through *sessions*, during which a client issues queries and retrieves the results in chunks of limited size [7]. Sessions are well isolated from each other, also concurrent modifications of the directory (i.e., new service registrations, updates, and removal of services from the directory) do not affect the sequence of results after a query has been issued. We have implemented a simple but very efficient concurrency control scheme which may delay the visibility of directory updates but does not require any synchronization activities within sessions. Hence, our concurrency control mechanism has no negative impacts on the scalability with respect to the number of concurrent sessions.

As in a large-scale directory the number of (partially) matching results for a query may be very high, it is crucial to order the result set within the directory according to heuristics and transfer first the better matches to the client. If the ranking heuristics work well, only a small part of the possibly large result set has to be transferred, thus saving network bandwidth and boosting the performance of a directory client that executes a service composition algorithm (the results are returned incrementally, once a result fulfills the client's requirements, no further results need to be transmitted). However, the ranking heuristics depend on the concrete composition algorithm. For each service composition algorithm (e.g., forward chaining, backward chaining, etc.), a different ranking heuristic may be better adapted. Because research on service composition is still in the beginning and the directory cannot anticipate the needs of all possible service integration algorithms, our directory supports *user-defined pruning and ranking functions*.

Custom pruning and ranking functions allow the execution of user-defined application-specific heuristics directly within the directory, close to the data, in order to transfer the best results for a query first. They dramatically increase the flexibility of our directory, as the client is able to tailor the processing of directory queries according to its needs. The pruning and ranking functions are written in Java and dynamically installed during service composition sessions by means of *mobile code*. Because the support of mobile code increases the vulnerability of systems and custom ranking functions may be abused for various kinds of attacks, such as denial-of-service attacks consuming a vast amount of memory and processing resources within the directory, our directory imposes severe restrictions on the code of these functions.

As the main contributions of this paper, we show how our directory supports user-defined pruning and ranking functions. We present the restricted programming model for pruning and ranking functions and discuss how they are used within directory queries. Moreover, we explain how our directory protects itself against malicious code. Performance evaluations point up the necessity to support heuristic pruning and ranking functions that are tailored to specific service composition algorithms.

This paper is structured as follows: In Section 2 we explain how service descriptions can be numerically encoded as sets of intervals and we give an overview of the index structure for multidimensional data on which our directory implementation is based. In Section 3 we show how the directory can be dynamically extended by user-defined pruning and ranking functions. We discuss some implementation details, in particular showing how the directory protects itself against malicious code. In Section 4 we present some experimental results that illustrate the need for dynamically installing application-specific, heuristic pruning and ranking functions. Section 5 presents ongoing research and our plans to improve the directory service in the near future. Finally, Section 6 concludes this paper.

2 Directories for Service Descriptions

Since we assume a large-scale open environment with a high number of available services, the integration process has to be able to discover relevant services incrementally through queries to the service directory. Interleaving the integration process with service discovery in a large-scale directory is a novelty of our approach. In this section we give an overview of the basic features of our directory and the used indexing structures.

2.1 Existing Directories

Currently, UDDI (Universal Description, Discovery, and Integration) [26] is the state-of-the-art for directories of web services. UDDI is an industrial effort to create an open specification for directories of service descriptions. It builds on existing technology standardized by the World Wide Web Consortium[3] like the eXtensible Markup Language

[3] `http://www.w3c.org/`

(XML), the Simple Object Access Protocol (SOAP), and the Web Services Description Language (WSDL). The UDDI standard is clear in terms of data models and query API, but suffers from the fact that it considers service descriptions to be completely opaque.

A more complex method for discovering relevant services from a directory of advertisements is matchmaking. In this case the directory query (requested capabilities) is formulated in the form of a service description template that presents all the features of interest. This template is then compared with all the entries in the directory and the results that have features compatible with the features of interest are returned. A good amount of work exists in the area of matchmaking, including LARKS [24] and the newer efforts geared towards OWL-S [21]. Other approaches include the Ariadne mediator [15].

2.2 Match Types

We consider four match relations between a query Q and a service S:

Exact: S is an exact match of Q.
PlugIn: S is a plug-in match for Q, if S could be always used instead of Q.
Subsumes: Q contains S. In this case S could be used under the condition that Q satisfies some additional runtime constraints such that it is specific enough for S.
Overlap: Q and S have a given intersection. In this case, runtime constraints both over Q and S have to be taken into account.

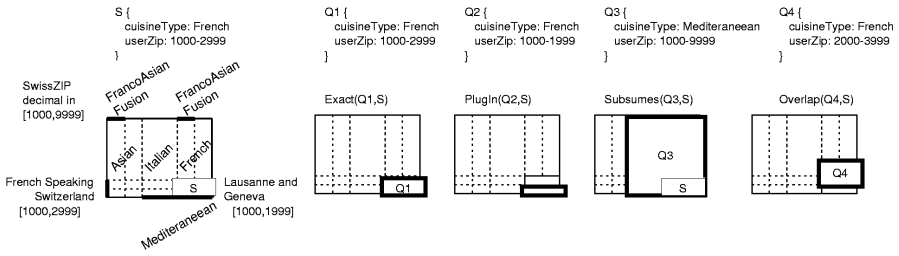


Fig. 1. Match types of inputs of query Q and service S by 'precision': *Exact*, *PlugIn*, *Subsumes*, *Overlap*.

In the example in Fig. 1 we show how the match relation is determined between the inputs available from the queries $Q1$, $Q2$, $Q3$, $Q4$ and the inputs required by the service S. We can order the types of match by 'precision' as follows: *Exact*, *PlugIn*, *Subsumes*, *Overlap*. We consider *Subsumes* and *Overlap* as 'partial' matches. The first three relations have been previously identified by Paolucci in [21] and the fourth, *Overlap*, was identified by Li [17] and Constantinescu [8].

Determining one match relation between a query description and a service description requires that subsequent relations are determined between all the inputs of the query Q and service S and between the outputs of the service S and query Q (note the reversed order of query and services in the match for outputs). Our approach is more complex than the one of Paolluci in that we take also into account the relations between the properties that introduce different inputs or outputs (equivalent to parameter names). This is important for disambiguating services with equivalent signatures (e.g., we can disambiguate two services that have two string outputs by knowing the names of the respective parameters).

2.3 Numerically Encoding Services and Queries

Service descriptions are a key element for service discovery and service composition and should enable automated interactions between applications. Currently, different overlapping formalisms are proposed (e.g., [29], [26], [11], [12]) and any single choice could be quite controversial due to the trade-off between expressiveness and tractability specific to any of the aforementioned formalisms.

In this paper we partially build on existing developments, such as [29], [1], and [11], by considering a simple table-based formalism where each service is described by a set of tuples mapping service parameters (unique names of inputs or outputs) to parameter types (the spaces of possible values for a given parameter). Parameter types can be expressed either as sets of intervals of basic data types (e.g., date/time, integers, floating-points) or as classes of individuals. Class parameter types can be defined in a descriptive language like XML Schema [30] or in the Ontology Web Language [28]. From the descriptions we can derive a directed graph (DG) of simple 'is-a' relations either directly or by using a description logic classifier.

For efficiency reasons, we represent the DG numerically. We assume that each class will be represented as a set of intervals. We encode each parent-child relation by subdividing each of the intervals of the parent; in the case of multiple parents the child class is represented by the union of the sub-intervals resulting from the encoding of each of the parent-child relations. Since for a given domain we can have several parameters represented by intervals, the space of all possible parameter values can be represented as a rectangular hyperspace with a dimension for each parameter. Details concerning the numerical encoding of services can be found in [8].

2.4 Multidimensional Access Methods – GiST

The need for efficient discovery and matchmaking leads to a need for search structures and indexes for directories. We consider numerically encoded service descriptions as multidimensional data and use techniques related to the indexing of such kind of information in the directory. This approach leads to local response times in the order of milliseconds for directories containing tens of thousands (10^4) of service descriptions.

The indexing technique that we use is based on the Generalized Search Tree (GiST), proposed as a unifying framework by Hellerstein [14] (see Fig. 2). The design principle of GiST arises from the observation that search trees used in databases are balanced trees with a high fanout in which the internal nodes are used as a directory and the leaf nodes point to the actual data. Each internal node holds a key in the form of a predicate P and a number of pointers to other nodes (depending on system and hardware constraints, e.g., filesystem page size). To search for records that satisfy a query predicate Q, the paths of the tree that have keys P satisfying Q are followed.

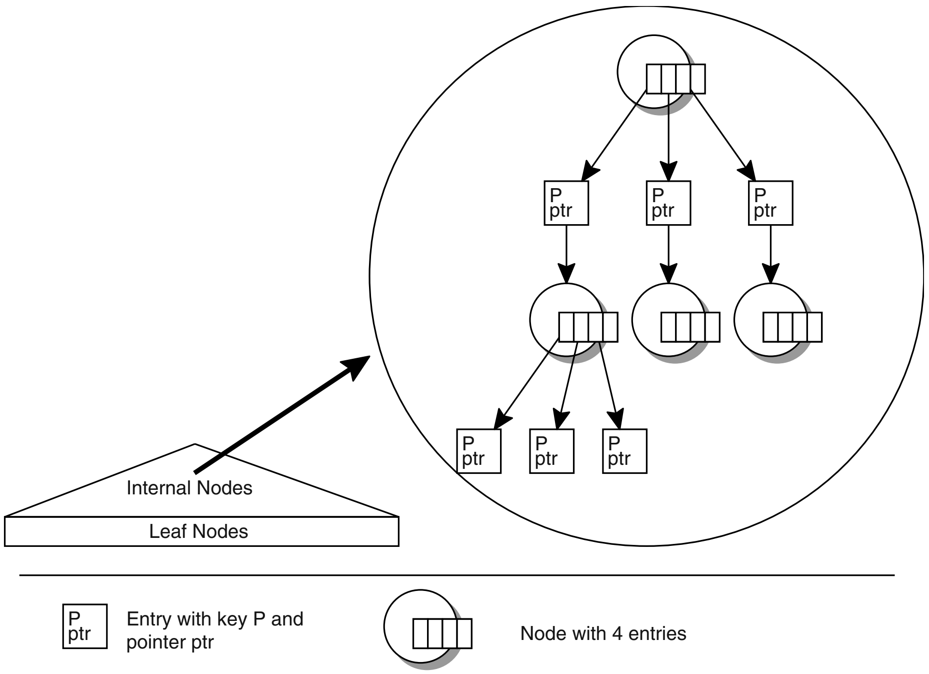


Fig. 2. Generalised Search Tree (GiST).

3 Custom Pruning and Ranking Functions

As directory queries may retrieve large numbers of matching entries (especially when partial matches are taken into consideration), our directory support sessions in order to incrementally access the results of a query [7]. By default, the order in which matching service descriptions are returned depends on the actual structure of the directory index (the GiST structure discussed before). However, depending on the service integration algorithm, ordering the results of a query according to certain heuristics may significantly improve the performance of service composition. In order to avoid the transfer of a large number of service descriptions, the pruning, ranking, and sorting according to application-dependent heuristics should occur directly within the directory. As for each service integration algorithm a different pruning and ranking heuristic may be better suited, our directory allows its clients to define custom pruning and ranking functions which are used to select and sort the results of a query. This approach can be seen as a form of remote evaluation [13]. Within a service integration session multiple pruning and ranking functions may be defined. Each query in a session may be associated with a different one.

3.1 API for Pruning and Ranking Functions

The pruning and ranking functions receive as arguments information concerning the matching of a service description in the directory with the current query. They return a value which represents the quality of the match. The bigger the return value, the better the service description matches the requirements of the query. The sequence of results as returned by the directory is sorted in descending order of the values calculated by the pruning and ranking functions (a higher value means a better match). Results for which the functions evaluate to zero come at the end, a negative value indicates that the match is too poor to be returned, i.e., the result is discarded and not passed to the client (pruning).

As arguments the client-defined pruning and ranking functions take four `ParamSet` objects corresponding to the input and output parameter sets of the query, and respectively of the service. The `ParamSet` object provides methods like size, membership, union, intersection, and difference (see Fig. 3). The size and membership methods require only the current `ParamSet` object, while the union, intersection, and difference methods use two `ParamSet` objects – the current object and a second `ParamSet` object passed as argument.

It is important to note that some of the above methods address two different issues at the same time:

1. Basic *set operations*, where a set member is defined by a parameter name and its type; for deciding the equality of parameters with same name and different types a user-specified expression is used.
2. *Computation of new types* for some parameters in the resulting sets; when a parameter is common to the two argument sets its type in the resulting set is computed with a user-specified expression.

The explicit behavior of the `ParamSet` methods is the following:

size: Returns the number of parameters in the current set.

containsParam: Returns true if the current set contains a parameter with the same name as the method argument (regardless of its type).

union: Returns the union of the parameters in the two sets. For each parameter that is common to the two argument sets the type in the resulting set is computed according to the user-specified expression `newTypeExpr`.

intersection: Returns the parameters that are common to the two sets *AND* for which the respective types conform to the equality test specified by the `eqTestExpr`. The type of these parameters in the resulting set is computed according to the user-specified expression `newTypeExpr`.

minus: Returns the parameters that are present only in the current set and in the case of common parameters only those that *DO NOT* conform to the equality test specified by the `eqTestExpr`. For the latter kind of parameters the type in the resulting set is computed according to the user-specified expression `newTypeExpr`.

Parameters whose `newTypeExpr` would be the empty type (called `NOTHING` in the table below) are removed from the resulting set.

The expressions used in the `eqTestExpr` and `newTypeExpr` parameters have the same format and they are applied to parameters that are common to the two `ParamSet` objects passed to a `union`, `intersection`, or `minus` method. For such kind of parameter we denote its type in the two argument sets as `A` and `B`. The expressions are created from these two types by using some extra constructors based on the Description Logic language OWL [28] like $\top$, $\bot$, $\neg$, $\sqcap$, $\sqcup$, $\sqsubseteq$, $\equiv$. The expressions are built by specifying a constructor type and which of the argument types `A` or `B` should be negated. For the single type constructors $\top$ and $\bot$ negation cannot be specified and for the constructors A and B the negation is allowed only for the respective type (e.g., for the constructor type A, only $\neg A$ can be set).

Constructor type	$\neg A$?	$\neg B$?	Possible expressions
`THING`	-	-	$\top$
`NOTHING`	-	-	$\bot$
`A`	Y/N	-	$A, \neg A$
`B`	-	Y/N	$B, \neg B$
`UNION`	Y/N	Y/N	$A \sqcup B$, $A \sqcup \neg B$, $\neg A \sqcup B$, $\neg A \sqcup \neg B$
`INTERSECTION`	Y/N	Y/N	$A \sqcap B$, $A \sqcap \neg B$, $\neg A \sqcap B$, $\neg A \sqcap \neg B$
`SUBCLASS`	Y/N	Y/N	$A \sqsupseteq B$, $A \sqsupseteq \neg B$, $\neg A \sqsupseteq B$, $\neg A \sqsupseteq \neg B$
`SUPERCLASS`	Y/N	Y/N	$A \sqsubseteq B$, $A \sqsubseteq \neg B$, $\neg A \sqsubseteq B$, $\neg A \sqsubseteq \neg B$
`SAMECLASS`	Y/N	Y/N	$A \equiv B$, $A \equiv \neg B$, $\neg A \equiv B$, $\neg A \equiv \neg B$

We represent an expression as a bit vector having a value corresponding to its respective constructor type. For encoding the negation of any of the types that are arguments to the constructor, two masks can be applied to the constructor types: `NEG_A` and `NEG_B`. For the actual encoding, see Fig. 3. For example, $A \sqcap \neg B$ will be expressed as `ParamSet.INTERSECTION | ParamSet.NEG_B`.

As an example of API usage, assume we need to select the parameters that are common to the two sets X and Y, which have in X a type that is more specific than the one in Y. In the result set we would like to preserve the type values in X. The following statement can be used for this purpose: `X.intersection(Y, ParamSet.SUPERCLASS, ParamSet.A)`.

The directory supports pruning and ranking functions written in a subset of the Java programming language. The code of the functions is provided as a compiled Java class. The class has to implement the `Ranking` interface shown in Fig. 3.

Processing a user query requires traversing the GiST structure of the directory starting from the root node. While `rankInner()` is invoked for inner nodes of the directory tree, `rankLeaf()` is called on leaf nodes. `rankLeaf()` receives as arguments `sin` and `sout` the exact parameter sets as defined by the service description stored in the directory. Hence, `rankLeaf()` has to return a concrete heuristic value for the given service description. In contrast, `rankInner()` receives as arguments `sin` and `sout` supersets of the input/output parameters found in any leaf node of its subtree. The type of each parameter is a supertype of the parameter found in any leaf node (which has the parameter) in the subtree. `rankInner()` has to return a heuristic value which is bigger or equal than all possible ranking values in the subtree. That is, for an inner node the heuristic function has to return an upper bound of the best ranking value that could be found in the subtree. If the upper bound of the heuristic ranking value in the subtree cannot be determined, `Double.POSITIVE_INFINITY` may be used.

The pruning and ranking functions enable the lazy generation of the result set based on a *best-first search* where the visited nodes of the GiST are maintained in a heap or

```
public interface Ranking {
  double rankLeaf(  ParamSet qin, ParamSet qout, ParamSet sin, ParamSet sout );
  double rankInner( ParamSet qin, ParamSet qout, ParamSet sin, ParamSet sout );
}

public interface ParamSet {
  static final int THING=1, NOTHING=2, A=3, B=4, UNION=5, INTERSECTION=6,
                   SUBCLASS=7, SUPERCLASS=8, SAMECLASS=9, NEG_A=16, NEG_B=32;

  int size();
  boolean containsParam( String paramName );
  ParamSet union( ParamSet p, int newTypeExpr );
  ParamSet minus( ParamSet p, int eqTestExpr, int newTypeExpr );
  ParamSet intersection( ParamSet p, int eqTestExpr, int newTypeExpr );
}
```

Fig. 3. The API for ranking functions.

priority queue and the most promising one is expanded. If the most promising node is a leaf node, it can be returned. Further nodes are expanded only if the client needs more results. This technique is essential to reduce the processing time in the directory until the the first result is returned, i.e., it reduces the response time. Furthermore, thanks to the incremental retrieval of results, the client may close the result set when no further results are needed. In this case, the directory does not spend resources to compute the whole result set. Consequently, this approaches reduces the workload in the directory and increases its scalability. In order to protect the directory from attacks, queries may be terminated if the size of the internal heap or priority queue or the number of retrieved results exceed a certain threshold defined by the directory service provider.

3.2 Exemplary Ranking Functions

In the example in Fig. 4 two basic ranking functions are shown, the first one more appropriate for service composition algorithms using forward chaining (considering only complete matches), the second for algorithms using backward chaining. Note the weakening of the pruning conditions for the inner nodes.

3.3 Safe and Efficient Execution of Ranking Functions

Using a subset of Java as programming language for pruning and ranking functions has several advantages: Java is well known to many programmers, there are lots of programming tools for Java, and, above all, it integrates very well with our directory service, which is completely written in Java.

Compiling and integrating user-defined ranking functions into the directory leverages state-of-the-art optimizations in recent JVM implementations. For instance, the HotSpot VM [23] first interprets JVM bytecode [18] and gathers execution statistics. If code is executed frequently enough, it is compiled to optimized native code for fast execution. In this way, frequently used pruning and ranking functions are executed as efficiently as algorithms directly built into the directory.

```
public final class RankingForward implements Ranking {
  public double rankLeaf( ParamSet qin, ParamSet qout,
                          ParamSet sin, ParamSet sout ) {
    // discard service if it requires parameters that are not in the query;
    // the provided input has to be more specific than the required one
    if (sin.minus(qin, ParamSet.SUBCLASS, ParamSet.A).size() > 0) return -1.0d;

    // services that provide more required parameters are better;
    // the provided output has to be more specific than the required one
    return (double)
             sout.intersection(qout, ParamSet.SUPERCLASS, ParamSet.A).size();
  }

  public double rankInner( ParamSet qin, ParamSet qout,
                           ParamSet sin, ParamSet sout ) {
    // for forward chaining, pruning inner nodes is not possible,
    // but an upper bound of the overlap of the outputs can be easily computed
    return (double)
             sout.intersection(qout, ParamSet.INTERSECTION, ParamSet.A).size();
  }
}

public final class RankingBackward implements Ranking {
  public double rankLeaf( ParamSet qin, ParamSet qout,
                          ParamSet sin, ParamSet sout ) {
    // discard service if it does not provide any required output
    if (sout.intersection(qout, ParamSet.SUPERCLASS, ParamSet.A).size() == 0)
      return -1.0d;

    // services that reduce most the number of required outputs are better
    ParamSet remaining = qout.minus(sout, ParamSet.SUBCLASS, ParamSet.A);
    ParamSet newRequired = sin.minus(qin, ParamSet.SUBCLASS, ParamSet.A);
    ParamSet required = remaining.union(newRequired, ParamSet.INTERSECTION);
    return 1 / (double)(1+required.size());
  }

  public double rankInner( ParamSet qin, ParamSet qout,
                           ParamSet sin, ParamSet sout ) {
    if (sout.intersection(qout, ParamSet.INTERSECTION, ParamSet.A).size() == 0)
      return -1.0d;

    ParamSet remaining = qout.minus(sout, ParamSet.INTERSECTION, ParamSet.A);
    return 1 / (double)(1+remaining.size());
  }
}
```

Fig. 4. Exemplary pruning and ranking functions.

The class containing the ranking function is analyzed by our special bytecode verifier which ensures that the user-defined ranking function always terminates within a well-defined time span and does not interfere with the directory implementation. Efficient, extended bytecode verification to enforce restrictions on JVM bytecode for the safe execution of untrusted mobile code has been studied in the JavaSeal [27] and in the J-SEAL2 [2,5] mobile object kernels. Our bytecode verifier ensures the following conditions:

– The `Ranking` interface is implemented.
– Only the methods of the `Ranking` interface are provided.

- The control-flow graphs of the `rankLeaf()` and `rankInner()` methods are acyclic. The control-flow graphs are created by an efficient algorithm with execution time linear with the number of JVM instructions in the method.
- No exception handlers (using malformed exception handlers, certain infinite loops can be constructed that are not detected by the standard Java verifier, as shown in [6]). If the ranking function throws an exception (e.g., due to a division by zero), its result is to be considered zero by the directory.
- No JVM subroutines (they may result from the compilation of `finally{}` clauses).
- No explicit object allocation. As there is some implicit object allocation in the set operations in `ParamSet`, the number of set operations and the maximum size of the resulting sets are limited.
- Only the interface methods of `ParamSet` may be invoked, as well as a well-defined set of methods from the standard mathematics package.
- Only the static fields defined in the interface `ParamSet` may be accessed.
- No fields are defined.
- No synchronization instructions.

These restrictions ensure that the execution time of the custom pruning and ranking function is bounded by the size of its code. Hence, an attacker cannot crash the directory by providing, for example, a pruning and ranking function that contains an endless loop. Moreover, these functions cannot allocate memory. Our extended bytecode verification algorithm is highly efficient, its performance is linear with the size of the pruning and ranking methods. As a prevention against denial-of-service attacks, our directory service allows to set a limit for the size of custom functions.

Pruning and ranking functions are loaded by separate classloaders, in order to support multiple versions of classes with the same name (avoiding name clashes between multiple clients) and to enable garbage collection of the class structures. The loaded class is instantiated and casted to the `Ranking` interface that is loaded by the system classloader. The directory implementation (which is loaded by the system classloader) accesses the user-defined functions only through the `Ranking` interface.

As service integration clients may use the same ranking functions in multiple sessions, our directory keeps a cache of ranking functions. This cache maps a hashcode of the function class to a structure containing the function bytecode as well as the loaded class. In case of a cache hit the user-defined function code is compared with the cache entry, and if it matches, the function in the cache is reused, skipping verification and avoiding to reload it with a separate classloader. Due to the restrictions mentioned before, multiple invocations of the same ranking function cannot influence each other. The cache employs a least-recently-used replacement strategy. If a function is removed from the cache, it becomes eligible for garbage collection as soon as it is not in use by any service integration session.

4 Evaluation

We have evaluated our approach by carrying out tests on random service descriptions and service composition problems that were generated as described in [9]. As we consider

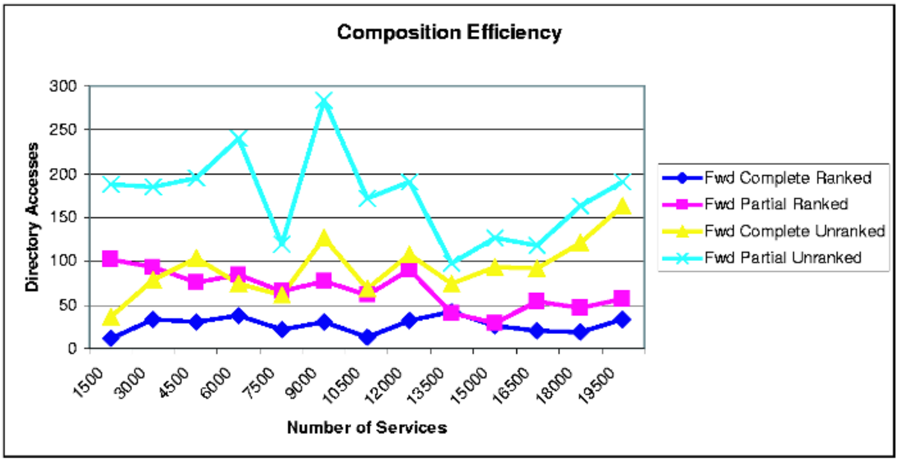


Fig. 5. Impact of ranking functions on composition algorithms.

directory accesses to be a computationally expensive operation we use them as a measure of efficiency.

The problems have been solved using two forward chaining composition algorithms: One that handles only complete type matches and another one that can compose partially matching services, too [10]. When running the algorithms we have used two different directory configuration: The first configuration was using the extensible directory described in this paper which supports custom pruning and ranking functions, in particular using the forward chaining ranking function described in Fig. 4. In the second configuration we used a directory which is not aware of the service composition algorithm (e.g., forward complete, backward, etc.) and cannot be extended by client code. This directory implements a generic ordering heuristic by considering the number of overlapping inputs in the query and in the service, plus the number of overlapping outputs in the query and in the service.

For both directories we have used exactly the same set of service descriptions and at each iteration we have run the algorithms on exactly the same random problems. As it can be seen in Fig. 5, using custom pruning and ranking functions consistently improves the performance of our algorithms. In the case of complete matches the improvement is up to a factor of 5 (for a directory of 10500 services) and in the case of partial matches the improvement is of a factor of 3 (for a directory of 9000 of services).

5 Future Work

As discussed in Section 3, currently the directory enforces severe restrictions on the pruning and ranking functions in order to protect itself against malicious or erroneous code. For instance, the code has to be sequential and it must not explicitly allocate memory.

We are trying to relax these restrictions with the aid of resource management mechanisms, granting a certain amount of memory and CPU cycles for the execution of the pruning and ranking function which will be stopped if they try to exceed their allowed quota. For this purpose we will use J-RAF2[4], The Java Resource Accounting Framework, Second Edition, which offers a fully portable resource management layer for Java [5,3, 4].

J-RAF2 rewrites the bytecode of Java classes before they are loaded and transforms the program in order to expose details concerning its resource consumption during execution. Currently, J-RAF2 addresses memory and CPU control. For memory control, object allocations are intercepted in order to verify that no memory limit is exceeded. For CPU control, the number of executed bytecode instructions are counted and periodically the system checks whether a running thread exceeds its granted CPU quota. J-RAF2 has been successfully tested in standard J2SE, J2ME, and J2EE environments. Due to special implementation techniques, the overhead for resource management is reasonably small, about 20–30%, and only the code rewritten for resource management (i.e., the custom pruning and ranking function) incurs this overhead.

Based on J-RAF2, we will be able to remove certain restrictions on the programming model of pruning and ranking functions. For instance, loops will be allowed in the control flow. Enriching the programming model will also allow the directory to expose a more flexible API to the custom pruning and ranking functions.

In addition to this security-related issue, we are working on a high-level declarative query language, which will ease the specification of query heuristics. We will use on-the-fly compilation techniques to generate Java bytecode for customizing the traversal of the directory tree.

Yet another issue we are currently working on is the distribution of the directory service within a network, in order to increase its availability and scalability and to prevent it from becoming a central point of failure and performance bottleneck.

6 Conclusion

Efficient service integration in an open environment populated by a large number of services requires a highly optimized interaction between large-scale directories and service integration engines. Our directory service addresses this need with special features for supporting service composition: indexing techniques allowing the efficient retrieval of (partially) matching services, service integration sessions offering incremental data retrieval, as well as user-defined pruning and ranking functions that enable the installation of application-specific ranking heuristics within the directory. In order to efficiently support different service composition algorithms, it is important not to hard-code ordering heuristics into the directory, but to enable the dynamic installation of specific pruning and ranking heuristics. Thanks to the custom pruning and ranking functions, the most promising results from a directory query are returned first, which helps to reduce the number of transferred results and to save network bandwidth. Moreover, the result set is generated lazily, reducing response time and the workload in the directory. As user-defined pruning and ranking functions may be abused to launch denial-of-service attacks

[4] `http://www.jraf2.org/`

against the directory, we impose severe restrictions on the accepted code. Performance measurements underline the need for applying application specific heuristics to order the results that are returned by a directory query.

References

1. D.-S. C. A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the Semantic Web. *Lecture Notes in Computer Science*, 2342, 2002.
2. W. Binder. Design and implementation of the J-SEAL2 mobile agent kernel. In *The 2001 Symposium on Applications and the Internet (SAINT-2001)*, San Diego, CA, USA, Jan. 2001.
3. W. Binder and V. Calderon. Creating a resource-aware JDK. In *ECOOP 2002 Workshop on Resource Management for Safe Languages*, Malaga, Spain, June 2002.
4. W. Binder and J. Hulaas. Self-accounting as principle for portable CPU control in Java. In *5th Annual International Conference on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays'2004)*, Erfurt, Germany, Sept. 2004.
5. W. Binder, J. Hulaas, A. Villazón, and R. Vidal. Portable resource control in Java: The J-SEAL2 approach. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA-2001)*, Tampa Bay, Florida, USA, Oct. 2001.
6. W. Binder and V. Roth. Secure mobile agent systems using Java: Where are we heading? In *Seventeenth ACM Symposium on Applied Computing (SAC-2002)*, Madrid, Spain, Mar. 2002.
7. I. Constantinescu, W. Binder, and B. Faltings. Directory services for incremental service integration. In *First European Semantic Web Symposium (ESWS-2004)*, Heraklion, Greece, May 2004.
8. I. Constantinescu and B. Faltings. Efficient matchmaking and directory services. In *The 2003 IEEE/WIC International Conference on Web Intelligence*, 2003.
9. I. Constantinescu, B. Faltings, and W. Binder. Large scale testbed for type compatible service composition. In *ICAPS 04 workshop on planning and scheduling for web and grid services*, 2004.
10. I. Constantinescu, B. Faltings, and W. Binder. Large scale, type-compatible service composition. In *IEEE International Conference on Web Services (ICWS-2004)*, San Diego, CA, USA, July 2004.
11. DAML-S. DAML Services, http://www.daml.org/services.
12. FIPA. Foundation for Intelligent Physical Agents Web Site, http://www.fipa.org/.
13. A. Fuggetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.
14. J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proc. 21st Int. Conf. Very Large Data Bases, VLDB*, pages 562–573. Morgan Kaufmann, 11–15 1995.
15. C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada. The Ariadne Approach to Web-Based Information Integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.
16. O. Lassila and S. Dixit. Interleaving discovery and composition for simpleworkflows. In *Semantic Web Services, 2004 AAAI Spring Symposium Series*, 2004.
17. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*, 2003.

18. T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, Reading, MA, USA, second edition, 1999.
19. S. McIlraith, T. Son, and H. Zeng. Mobilizing the semantic web with daml-enabled web services. In *Proc. Second International Workshop on the Semantic Web (SemWeb-2001)*, Hongkong, 2001.
20. S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, Apr. 22–25 2002. Morgan Kaufmann Publishers.
21. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, 2002.
22. S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *In 11th World Wide Web Conference (Web Engineering Track)*, 2002.
23. Sun Microsystems, Inc. Java HotSpot Technology. Web pages at `http://java.sun.com/products/hotspot/`.
24. K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the internet. In *Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*, Stanford University, USA, March 1999.
25. S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In *Proceeding of the AAAI-2002 Workshop on Intelligent Service Integration*, pages 1–7, Edmonton, Alberta, Canada, July 2002.
26. UDDI. Universal Description, Discovery and Integration Web Site, http://www.uddi.org/.
27. J. Vitek, C. Bryce, and W. Binder. Designing JavaSeal or how to make Java safe for agents. Technical report, University of Geneva, July 1998.
28. W3C. OWL web ontology language 1.0 reference, http://www.w3.org/tr/owl-ref/.
29. W3C. Web services description language (wsdl) version 1.2, http://www.w3.org/tr/wsdl12.
30. W3C. XML Schema, http://www.w3.org/xml/schema.
31. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, 2003.

Specification and Execution of Declarative Policies for Grid Service Selection*

Massimo Marchi[1], Alessandra Mileo[2], and Alessandro Provetti[3]

[1] DSI - Università degli studi di Milano
Milan, I-20135 Italy
marchi@dsi.unimi.it.

[2] DICo - Università degli studi di Milano
Milan, I-20135 Italy
mileo@dico.unimi.it.

[3] Dip. di Fisica - Università degli studi di Messina
Messina, I-98166 Italy
ale@unime.it

Abstract. We describe a modified Grid architecture that allows to specify and enforce connection policies with preferences and integrity constraints. This is done by interposing a policy enforcement engine between a calling application and the relative client stubs. Such policies are conveniently expressed in the declarative policy specification language PPDL. In particular, PPDL allows expressing preferences on how to enforce constraints on action execution. PPDL policies are evaluated by translating them into a Logic Program with Ordered Disjunctions and calling the psmodels interpreter. We describe an experimental architecture that enforces connection policies by catching and filtering only service requests. The process is completely transparent to both client applications and Grid services. There are clear advantages in having the connection logic expressed declaratively and externally to applications.

Keywords: Grid Services. Customization. User preferences. Declarative Policies. Answer Set Programming.

1 Introduction

In this article we how the standard Grid service architecture can be improved by interposing a policy enforcement engine between a calling application and the relative client stubs. Our policies can specify, among others, preferences and prohibitions in the *routing* of remote invocations to Web services (WS).

* This work has been supported by i) MIUR COFIN project *Formal Languages and Automata: Methods, Models and Applications* and ii) the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project. The authors participate in M^2AG: Milan-Messina actions group: http://mag.dsi.unimi.it/

Therefore, with our solution WS preference and invocation is not hard-coded into client applications but (declaratively) defined and enforced outside the clients, so that they can be (de)activated and modified at runtime. Our architecture is implemented so as remain transparent to both client application[1] and the invoked Web service.

The *Policy Description Language with Preferences* (PPDL), which is formally described below, is a recent development of the PDL language. PDL is a result of applied research at Bell Labs [9,12] on automated tools for network administration. PDL policies are high-level, i.e., abstract from the device they are applied to. Even though PPDL has rather simple constructs and is prima-facie a less expressive language than those traditionally considered in knowledge representation (Logic Programming, Description logics and others), it allows capturing the essence of routing control and allows us to keep the so-called *business logic* outside the code; so it can be inspected and changed any time transparently from the applications, which won't need rewriting. Finally, by adopting PPDL we keep policies in a declarative, almost documentation form yet with automated and relatively efficient enforcement. Performance is the result of adopting Answer Set Programming solvers [16] to execute the policy evaluation.

2 The Grid Service Architecture

Web services is a distributed technology that permits, in a worldwide network environment, to build effective client/server applications. A set of well-defined protocols, built mainly on XML and *Uniform Resource Identifier*(URI), is used to describe, address and communicate with services, thus achieving a large interoperability between different client and server implementations.

A typical WS may be viewed as a service dispenser (please see Figure 1 below). A generic *client* application, can consult a directory of available services, called Universal Description, Discovery and Integration (UDDI) Registry, invoke one of such services and retrieve the results in a fashion similar to that of usual Web sessions.
While we cannot dwell on the details of the WS architecture, let us just notice that each WS can be addressed by an URI. For our purposes, we will consider URIs that are simple URLs. The *Web Service Description Language* (WSDL) is used to describe how to communicate with WS, e.g., the format that service requests should have, that of service responses, the possible error messages and so on.

3 Our Experimental Architecture

In our experimental architecture we adopt *Grid Services,* an extension of Web Services available in the *Globus ToolKit 3 Framework* [11]. Grid services provide

[1] So far, however, we have considered only Java applications.

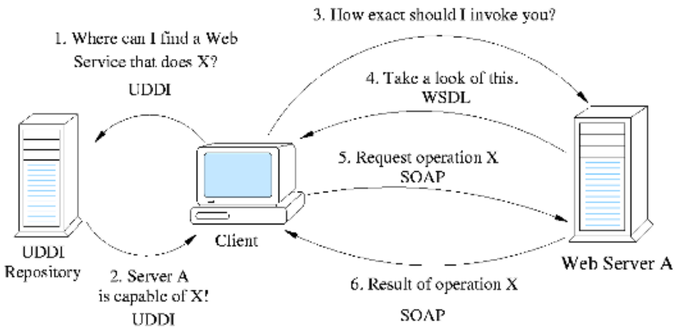

1. Where can I find a Web Service that does X? UDDI
2. Server A is capable of X! UDDI
3. How exact should I invoke you?
4. Take a look of this. WSDL
5. Request operation X SOAP
6. Result of operation X SOAP
UDDI Repository
Client
Web Server A

Fig. 1. The standard Grid architecture.

some graceful features not always supported by general Web Services, such as dynamic instance service creation, lifetime management and notification.

Typically, communication between client and server is made through a coupled object: client stub and server stub, that hides all low-level communication activity. Starting from WSDL service description, it is possible to automatically generate the code for client and server stubs. The policy module, which will be described in detail below, is inserted in the architecture by modifying the class that implements the client stub interface (see Figure 2 below).

In order to use a WS, a client application must go through two steps, which are now described in some detail.

In step 1, the application creates a *handler* for managing communication with the chosen service. Such handlers are in fact instances of the Java class that implements the so-called *client stubs*. For each service hosted by a given server an instance must be created that represents the service toward client applications. In fact, each service is addressed by an URI which reads something like *http://server.domain/Service/Math*. Such URI says that server *server.domain* is hosting service *Service/Math*.

In step 2, the client application actually calls the service by invoking the corresponding instance and passing all the call arguments. The called instance performs all the needed operation to communicate with the service, retrieves the results and return them to the client.

Our policy module enters into play at step 1, where it catches all creations and keeps a look-up table containing all available services. At that stage, URIs are translated into corresponding instance handlers during communication between policy module and client application. Moreover, the policy module catches all service calls, it enforces the policy by invoking the external psmodels solver and finally routes the call according to the policy results, thus achieving goals such as reliability, load balancing etc.

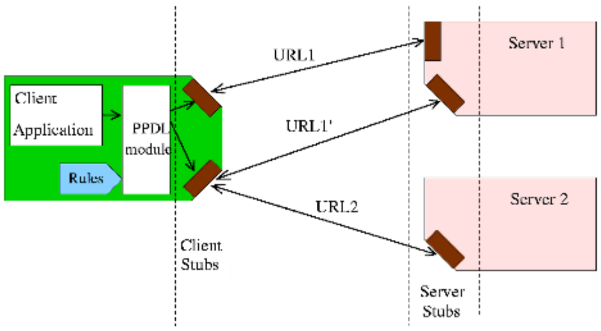


Fig. 2. PDL module location on WS client.

4 Introduction to PPDL

Developing and analyzing policies made of several provisions can be quite a complex tasks, in particular if one wants to ensure some sort of *policy-wide consistency*, that is for example, that no two different provisions of the policy result in conflicting actions to be executed. In order to guarantee *policy consistency*, declarative approaches to policy specification appear to be a promising viable solution.

One such approach to specify policies in network context has been recently proposed by Chomicki, Lobo, Naqvi [7,8,9] with the *Policy Description Language* (PDL). In that context, a (device-independent) policy is a description of how events received over a network (e.g., queries to data, connection requests etc.) are served by some given network terminal or *data server.* PDL allows managers to specify policies independently from the details of the particular device executing it. This feature is a great advantage when dealing with heterogeneous networks, which is most often the case nowadays. We refer the reader to works by Chomicki et al. [9] for a complete introduction and motivation for PDL in network management. In order to introduce our PPDL, we will now give an overview of its parent language PDL.

4.1 Overview of PDL

PDL can be described as an evolution of the *Event-Condition-Action* (ECA) schema of active databases. A PDL program is defined as a set of policy rules P_i and a consistency maintenance mechanism called *monitor*, composed by a set of rules M_i of the form

$$\begin{aligned} P_i :\ & e_1, \ldots e_m \ \mathbf{causes}\ a\ \mathbf{if}\ C \\ M_i :\ & \mathbf{never}\ a_1,\ \ldots,\ a_n\ \mathbf{if}\ C' \end{aligned}$$

where C, C' are Boolean conditions, $e_1, \dots e_m$ are events, which can be seen as input requests[2] and a is an action, which is understood to be a configuration command that can be executed by the network manager and actions $a_1 \dots a_n$ of M_i are forbidden from executing *simultaneously.*

PDL assumes that events and actions are syntactically disjoint and that rules are evaluated and applied in parallel. One may notice the lack of any explicit reference to time. In fact, PDL rules are interpreted in a discrete-time framework as follows. If at a given time t the condition is evaluated true and all the events have been received from the network, then at time $t+1$ action a is executed. As a result, we can see PDL policies as describing a *transducer.*

If the application of policies yields a set of actions that violates one of the rules in the monitor then the PDL interpreter will *cancel* some of them, but notice that selection of a particular action(s) to drop cannot be specified by the language as is. However, Chomicki et al. describe two general solutions, called action-cancellation and event-cancellation, respectively.

The declarative semantics of PDL policies is given by means of translation into Answer Set Programming (ASP), namely in the expressive framework of *disjunctive logic programs.* Also, thanks to that translation one can actually *run* a PDL policy against a set of input events by feeding the translated version to an ASP solver (see [16]) and inspecting the computed answer sets to find the actions dictated by the policy. In our language PPDL we retain and extend Chomicki et al. translation to get the same appealing features of a concise declarative semantics and interpretation via ASP solvers.

4.2 PPDL: Policy Description Language with Preferences

It should be observed that in PDL it is possible to specify which actions cannot execute together but it is not possible to specify what should be done in order to avoid violations. In other words, the administrator cannot specify which actions should preferentially be dropped, and what actions should be preferentially executed even in case of a violation. Indeed, in PDL the choice of which action to drop is non-deterministic.

We believe that flexible policy languages, by which one can specify whether and how to enforce constraints, are required. We have moved closer to achieve such result by defining an extension of PDL [2,1] that allows users to express preferences. This is done by reconstructing Brewka's ordered disjunction connective [4] into PDL, thus obtaining an output based on degrees of satisfaction of a preference rule.

The resulting language is called *PPDL: PDL with Preferences* and it enables users to specify preferences in policy enforcement (cancellation of actions) To

[2] Also, non-occurrence of an event may be in the premise of the rule. To allow for that, for each event e a dual event $\overline{e}$ is introduced, representing the fact that e has not been recorded. This is called *negation as failure*(NAF) and it is different than asserting $\neg e$, which means that an event corresponding to the negation of e has been recorded. In this paper we will not consider negated events.

describe a preference relation on action to be blocked when a constraint violation occur, we introduced constraints with the following syntax:

$$\textbf{never } a_1 \; \times \ldots \; \times \; a_n \textbf{ if } C. \tag{1}$$

which means that actions $a_1, \ldots, a_n$ cannot be executed together *and* –in case of constraint violation– a_1 should be blocked. If this is not possible (i.e. a_1 must be performed), block a_2, else block a_3 etc.; if all of $a_1, \ldots, a_{n-1}$ must be executed, then block a_n.

PPDL policies receive a declarative semantics and are computed by translating them into Brewka's Logic Programs with Ordered Disjunctions (LPODs). Ordered disjunctions are a relatively recent development in reasoning about preferences with Logic Programming and are subject of current work by [4,5], [6], [15] and others. One important aspect of Brewka's work is that preferred answer sets need not be minimal. This is a sharp departure from traditional ASP and in [1] we have investigated how adding preferences to PDL implies a trade-off between user-preferences and minimality of the solutions. Notice that, both in PDL and PPDL translations to ASP, minimality of answer sets corresponds to *minimality of the set of actions that get canceled in case of violations.*

4.3 Overview of LPOD

As mentioned earlier, Logic Programs with Ordered Disjunctions have been introduced by [4] in his work on combining Qualitative Choice Logic and Answer Set Programming. A new connective, called *ordered disjunction* and denoted with "×," is introduced. An LPOD consists of rules of the form

$$C_1 \; \times \; \ldots \; \times \; C_n \; \text{:--} \; A_1, \; \ldots, \; A_m, \; not \; B_1 \; \ldots, \; not \; B_k. \tag{2}$$

where the C_i, A_j and B_l are ground literals. The intuitive reading [4] of the rule (2) is:

> when $A_1, \ldots, A_m$ are observed and $B_1, \ldots, B_k$ are not observed, then
> if possible deduce C_1, but if C_1 is not possible, then deduce C_2,
> ...
> if all of $C_1, \ldots, C_{n-1}$ are not possible, then deduce C_n instead.

The × connective is allowed to appear in the head of rules only; it is used to define a preference relation so as to *select* some of the answer sets of a program by using ranking of literals in the head of rules, on the basis of a given strategy or context. The answer sets of a LPODs program are defined by Brewka as sets of atoms that maximize a preference relation induced by the "×-rules" of the program. Before describing the semantics, let us consider a simple example.

Example 1. (from [5]) Consider the Linux configuration domain, and the process of configuring a workstation. There might be several kinds of different preference criteria. First, there are usually several available versions for any given software package. In most cases we want to install the latest version, but sometimes, we have to use an older one. We can handle these preferences by defining a new atom for each different version and then demanding that at least one version should be selected if the component is installed. Second, a component may have also different variants (e.g. a normal version and a developer version). A common user would prefer to have the normal variant while a programmer would prefer the developer version. Suppose there are three versions of emacs available. This preferences can be modeled using rules expressed by LPODs syntax:

1. $emacs - 21.1 \ \times \ emacs - 20.7.2 \ \times \ emacs - 19.34 \ \ {:-}\ installed - emacs.$
2. $dev - library \ \times \ usr - library \ \ {:-}\ need - library, \ developer.$
3. $usr - library \ \times \ dev - library \ \ {:-}\ need - library, \ not \ developer.$

4.4 The Declarative Semantics of LPODs

The semantics of LPOD programs is given in terms of a model preference criterion over answer sets. [4] shows how Inoue and Sakama's split program technique can be used to generate programs whose answer sets characterize the LPOD preference models. In short, a LPOD program is rewritten into several split programs, where only one head appears in the conclusion. Split programs are created by iterating the substitution of each LPOD rule (2) with a rule of the form:

$$C_i \ {:-} A_1, \ldots, A_m, not \ B_1, \ldots, not \ B_k, not \ C_1, \ldots, not \ C_{i-1} \tag{3}$$

Consequently, Brewka defines answer sets for the LPOD program Π as the answer sets of any of the split programs generated from Π.

There is one very important difference between Gelfond and Lifschitz's answer sets and LPOD semantics: in the latter (set-theoretic) minimality of models is not always wanted, and therefore not guaranteed. This can be better explained by the following example.

Example 2. Consider these two facts:

1. $A \ \times \ B \ \times \ C.$
2. $B \ \times \ D.$

To *best satisfy* both ordered disjunctions, we would expect $\{A, \ B\}$ to be the single preferred answer set of this LPOD, even if this is not even an answer set of the corresponding disjunctive logic program (where "$\times$" is replaced by "$\vee$"). Indeed, according to the semantics of [10] $\{B\}$ satisfies both disjunctions and is *minimal.*

To sum it up, since minimality would preclude preferred answer sets to be considered dealing with preferences implies adopting non-minimal semantics.

LPOD programs are be interpreted by a special version of the solver *Smodels,* called *Psmodels,* which is presented in [5]. In a nutshell, LPOD programs are translated (by the *lparse* parser) into equivalent (but longer) ASP programs and then sent to *Psmodels.*
Now, we can go back to policies and describe how PPDL is mapped into LPOD.

4.5 Translating PPDL Policies into Answer Set Programming

Starting from a set of preference cancellation rules (1) we define LPOD *ordered blocking rules* as follows:

$$block(a_1) \;\times\; \ldots \;\times\; block(a_n) \;\;{:}{-}\; exec(a_1), \ldots,\; exec(a_n),\; C. \tag{4}$$

Since the PPDL-to-LPOD translation described above does not provide a mechanism for avoiding action block, the resulting program is deterministic: we would obtain answer sets where the leftmost action of each rules of the form (4) that fires is always dropped.

As a result, in [1] we argued that a simplified version of rule (4) can be formulated as follows:

$$block(a_1) \;\;{:}{-}\; exec(a_1), \ldots,\; exec(a_n),\; C. \tag{5}$$

This translation realizes a simple, deterministic preference criteria in canceling action violating a constraint, according to the given strategy: for each constraint, we put as leftmost action an action that shall always be dropped.

However, ordered disjunctions are appealing precisely when some actions *may not be blocked*. This can be specified by using a new rule called *anti-blocking rule* which is added to the language.

Anti-blocking rules. This rules allow users to describe actions that *cannot be filtered* under certain conditions. The syntax of *anti-blocking rule* is as follows:

$$\textbf{keep}\ a\ \textbf{if}\ C. \tag{6}$$

where a is an action that cannot be dropped when the boolean condition C is satisfied. This rule is applied whenever a constraint of the form (1) is violated, and a is one of the conflicting actions. In ASP, anti-blocking rules are mapped in a constraint formulated as follows:

$${:}{-}\; block(a),\; C. \tag{7}$$

which is intended as *action a cannot be blocked if condition C holds*. Notice that if we want to control the execution of action a, postulating that under condition C action a is executed *regardless*, then we should write, in PPDL:

$$\begin{array}{l} \emptyset\ \textbf{causes}\ a\ \textbf{if}\ C. \\ \textbf{keep}\ a\ \textbf{if}\ C. \end{array}$$

that will be translated in LPOD as follows:

$$\begin{array}{l} exec(a)\;\;{:}{-}\; C. \\ {:}{-}\; block(a),\; C. \end{array}$$

Unlike in traditional PDL, where actions are strictly the consequence of events, by the **causes** described above we allows *self-triggered* or *internal* actions. We should mention that, even without internal events, a PPDL policy with monitor, blocking and anti-blocking rules, may be inconsistent. Consider the following example referred to allocation of resource $res1$ among two different users $usr1$ and $usr2$.

Example 3. Take policy P_{res}:

$$\begin{aligned} P_{res} \; = \{ \; & need\_usr1\_res1 \ \textbf{causes} \ assign\_usr1\_res1. \\ & need\_usr2\_res1 \ \textbf{causes} \ assign\_usr2\_res1. \; \} \end{aligned}$$

and a preference monitor M_{res} saying that resource $res1$ cannot be assigned both to $usr1$ and $usr2$. In particular, it is preferable to drop the request of $usr2$, supposed he/she is less important than $usr1$. Moreover, if one of the users has an urgent need, than his/her request should not be dropped.

$$\begin{aligned} M_{res} \; = \{ \; & \textbf{never} \ assign\_usr1\_res1 \ \times \ assign\_usr2\_res1. \\ & \textbf{keep} \ assign\_usr1\_res1 \ \textbf{if} \ urgent\_usr1. \\ & \textbf{keep} \ assign\_usr2\_res1 \ \textbf{if} \ urgent\_usr2. \; \} \end{aligned}$$

where $urgent\_usr1$ and $urgent\_usr2$ stand for Boolean conditions. Both P_{res} and M_{res} are translated the following LPOD, named π_{res}:

$$\begin{aligned} & exec(assign\_usr1\_res1) \ \text{:--} \ occ(need\_usr1\_res1). \\ & exec(assign\_usr2\_res1) \ \text{:--} \ occ(need\_usr2\_res1). \\ & block(assign\_usr2\_res2) \times \ block(assign\_usr2\_res1) \ \text{:--} \ exec(assign\_usr1\_res1), \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad exec(assign\_usr2\_res1). \\ & \text{:--} \ block(assign\_usr1\_res2), urgent\_usr1. \\ & \text{:--} \ block(assign\_usr2\_res2), urgemt\_usr2. \end{aligned}$$

Now, suppose that events $need\_usr1\_res1$ and $need\_usr2\_res1$ has occurred. It is clear that if both the clients have urgent requests, π_{res} is inconsistent so the policy+monitor application yields an error and the requests should be transmitted again.

The simple example above shows that if we want to use prioritized semantics in extended PPDL, we have to be careful in introducing anti-blocking rules, in order to ensure that at least one action can be blocked whenever a constraint is violated.

5 The PPDL Specification of Grid Service Selection

This section gives a complete example of a Grid service scenario based on our architecture. In our Department there are three servers that implement grid services. Here we consider a grid service called *math* available on all three servers. The *math* service consists, essentially, of arithmetic functions. Clearly, we get the exact same service from all services, even though the implementation can vary to i) optimize performance on certain inputs and ii) adapt to the particular platform where the service is run.

In our scenario, several details regarding location and interface of the service are known and are made available for policy enforcement through tables. The following lookup table[3] is an example of a PPDL specification of the services we have access to:

Table 1. A service lookup table

URL	service
mag.usr.dsi.unimi.it/math	mag.math
zulu.usr.dsi.unimi.it/math	mag.math
grid001.usr.dsi.unimi.it/math	mag.math
grid002.usr.dsi.unimi.it/math	mag.math

In the context of the lookup table above, we have designed the PPDL policy described next: The goal is is to maximize computation per time unit, while keeping into account the sharp differences in performance among our servers.

P_1: req(I,M,L1,L2) **causes** send(Url,I,M,L1,L2)
if table(Url,I), M$\neq$m-plus.
P_2: req(I,M,L1,L2) **causes** send(Url,I,m-plus,L1,L2)
if table(Url,I), M=m-plus, L1$\leq$10, L2$\leq$10.
P_3: request(I,M,L1,L2) **causes** send(Url,I,m-fastplus,L1,L2)
if table(Url,I), M=m-plus, $L1 > 10$.
P_4: request(I,M,L1,L2) **causes** send(Url,I,m-fastplus,L1,L2)
if table(Url,I), M=m-plus, $L2 > 10$.

M_1: **never** send(grid001,I,M,L1,L2) $\times$ send(grid002,I,M,L1,L2)
if M=m-plus.
M_2: **never** send(zulu,I,M,L1,L2) **if** M=m-fastplus.
M_3: **never** send(grid002,I,M,L1,L2) $\times$ send(grid001,I,M,L1,L2)
if M=m-fastplus, $L1 > 20$.
M_4: **never** send(grid002,I,M,L1,L2) $\times$ send(grid001,I,M,L1,L2)
if M=m-fastplus, $L2 > 20$.
M_5: **never** send(grid001,I,M,L1,L2) $\times$ send(grid002,I,M,L1,L2)
if M=m-fastplus, L1$\leq$20, L2$\leq$20.

Rule P_1 simply says that an invocation of a method other than *m-plus* method, is sent to Web service where, according to the lookup table, such service is available. Policy rules P_2 to P_4 practically define the method *m-plus* and say that for such method, if at least one parameter is greater than 10, then a faster method called *m-fastplus* should be (transparently) invoked.

Monitor rules M_1 to M_5 tell how routing should be preferably performed according to the size of the parameters and the computational power of the

[3] There are several ways for creating the lookup table. For instance, it may be obtained by consulting the UDDI directory on the Web.

server. In particular, M_1 says that the *m-plus* method invocation should be blocked on server *grid001* with higher preference with respect to *grid002*, as the first one is faster than the second one, and we want it not to be busy with simple computation. M_2 prevents the client from sending a *m-fastplus* method invocation to the slow *zulu* server.

Rules M_3 to M_5 tell the client how to route *m-fastplus* method invocation among the faster servers, according to the size of the parameter: if both the parameters are less or equal to 20, a method invocation is send to *grid002*. Otherwise, it is routed to *grid001*, which is supposed to perform better with high values of the parameters.

5.1 The Software Layers

In general, a PPDL policy specification can be *animated* by the following step-by-step procedure outlined in Figure 3) below.

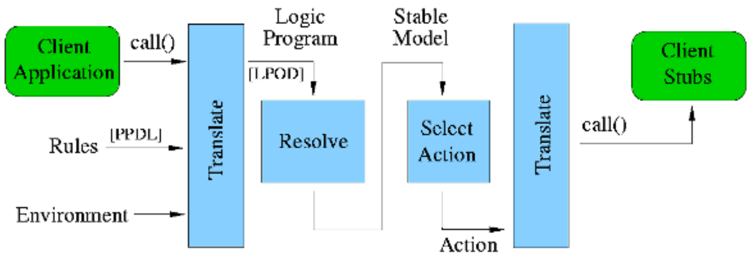


Fig. 3. The software layers of our architecture

First, the PPDL policy is translated into an Answer Set Program, following the encoding defined in [2]. Second, the resulting ASP program is fed to a solver that computes one of its answer sets. These answer set will contain, among other uninteresting atoms, a set of instances of the *execute(a_i)* predicate that describe the actions that should be executed next. An extractor takes the ASP solver output and extracts the $a_1, \ldots a_n$ actions to be executed, then it examines the action name and calls the appropriate routines, that will invoke the chosen client stub.

5.2 The Complete Architecture

As we have mentioned above, the architecture in Figure 4 is obtained by modifying the GT3 class, *ServiceLocator*, that creates an object instance for each client-grid connection. Each service is identified by an URL and provides a set of operation, or *methods*. In our architecture, the *Trapper* routine described in Figure 4 catches all outgoing calls made by the client application.

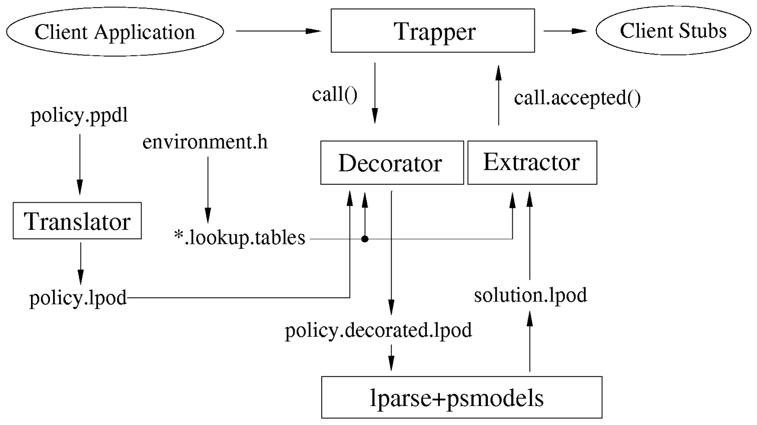


Fig. 4. The standard Grid architecture.

The *Trapper* method stores the URLs of available services in a lookup table and use them to pass from the Java object that represent the stub to the relative URL and vice versa. When the client application perform a method invocation, *Trapper* extract from that call i) the URL of the service, ii) the requested interface and method and iii) the arguments that should be passed to the remote method.

Next, Trapper translates all real names to symbolic values used in the PPDL policy. In our solution this step is performed by means of an external environment specification file, *environment.h.*

Now, the PPDL policy specification, *policy.ppdl,* needs to be translated to an LPOD program in order to apply it. This step is performed by *Translator.*

Next, *Decorator* assembles all call data, the policy and environment specifications together into a complete LPOD program. An external module, the *lparse+psmodels* box seen in Figure 4, interprets this program and extracts one (or more) answer set. The answer set contains the a set *accept* atoms describing executable, non-blocked actions.

Extractor extracts from the solution a subset of non-redundant actions by non-deterministically choosing an action from tie-breaks. Finally *Trapper* translates back the solution into a real client-stub call.

6 Conclusions and Open Problems

In this article we have described a new, experimental-yet-functional Grid service architecture that, in our opinion, has several advantages, thanks to having the *connection logic* expressed outside the application and in declarative format.

Our solution is transparent to the standard Grid service architecture and can be described as bringing to Grid services the same advantages that triggers and constraints bring to relational databases and their client applications.

Our implementation of the architecture is still in its infancy and several of the software layers may be improved with more sophisticated implementation and optimization. However, automatic mapping from PPDL to LPOD has already been described in details, and it is rather straightforward. We are working on a simple user-friendly interface to help users in writing and compiling their PPDL policies. Meanwhile, our experiments are suggesting that one important issue that needs to be investigated further is related to method calls routing when multiple solution are obtained.

The PPDL module may return different routing possibilities[4] for each method call, all this solution being equally preferred according to the PPDL semantics. Only one server invocation should be done for each call. In our prototype the one invocation to execute is chosen non-deterministically among all the possible ones. Clearly, when several parallel method calls are requested by the client application, it is important to have a method to distribute the calls among all available servers according to some criteria, e.g., performance improvement, overload avoidance or reliability. This may be done by using some planning techniques or by defining a further (internal) level of policy specification.

Acknowledgments. Thanks to A. Bertoni, E. Bertino, G. Gonzales and M. Ornaghi for useful discussions on this subject.

References

1. Bertino, E., Mileo, A. and Provetti, A., 2003. User Preferences VS Minimality in PPDL. In Buccafurri F. (editor), Proc. of *AGP03,* APPIA-GULP-PRODE. Available from *http://mag.dsi.unimi.it/PPDL/*
2. Bertino E., Mileo A. and Provetti A., 2003. *Policy Monitoring with User-Preferences in PDL.* Proc. of *NRAC 2003* IJCAI03 Workshop on Reasoning about Actions and Change.
   Available from *http://mag.dsi.unimi.it/PPDL/*
3. Bertino, E., Mileo, A. and Provetti, A., 2003. PDL with Maximum Consistency Monitors. Proc. of *Int'l Symp. on Methodologies for Intelligent Systems (ISMIS03).* Springer LNCS. Available from *http://mag.dsi.unimi.it/PPDL/*
4. Brewka, G., 2002. *Logic Programming with Ordered Disjunction.* Proc. of AAAI-02. Extended version presented at NMR-02.
5. Brewka, G., Niemelä I and Syrjänen T., 2002. *Implementing Ordered Disjunction Using Answer Set Solvers for Normal Programs.* Proc. of JELIA'02. Springer Verlag LNAI.
6. Buccafurri F., Leone L. and Rullo P., 1998. Disjunctive Ordered Logic: Semantics and Expressiveness. Proc. of KR'98. MIT Press, pp. 418-431.

[4] Notice that each possibility is represented by an *accept(...)* atom.

7. Chomicki J., Lobo J. and Naqvi S., 2000. *A logic programming approach to conflict resolution in policy management.* Proc. of KR2000, 7th Int'l Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufmann, pp 121–132.
8. J. Chomicki, J. Lobo, 2001. *Monitors for History-Based Policies.* Proc. of Int'l Workshop on Policies for Distributed Systems and Networks. Springer, LNCS 1995, pp. 57–72.
9. Chomicki J., Lobo J. and Naqvi S., 2003. *Conflict Resolution using Logic Programming.* IEEE Transactions on Knowledge and Data Engineering 15:2.
10. Gelfond, M. and Lifschitz, V., 1991. Classical negation in logic programs and disjunctive databases. New Generation Computing: 365–387.
11. Web location related to Web Services technologies.
    Globus Toolkit Framework: *http://www.globus.org/*
    World Wide Web Consortium: *http://www.w3c.org/*
12. Lobo J., Bhatia R. and Naqvi S., 1999. A Policy Description Language, in *AAAI/IAAI, 1999*, pp. 291–298.
13. Marchi M., Mileo A. and Provetti A., 2004. *Specification and execution of policies for Grid Service Selection.* Posters at ICWS2004 conference. IEEE press. Available from *http://mag.dsi.unimi.it/PPDL/*
14. Marchi M., Mileo A. and Provetti A., 2004. *Specification and execution of policies for Grid Service Selection.* Poster at Int'l Conference on Logic Programming (ICLP04) Spinger LNCS. Available from *http://mag.dsi.unimi.it/PPDL/*
15. Schaub T., and Wang K., 2001. *A comparative study of logic programs with preference.* Proc. of Int'l. Joint Conference on AI, IJCAI-01.
16. Web location of the most known ASP solvers.
    Aspps: *http://cs.engr.uky.edu/ai/aspps/*
    CMODELS: *http://www.cs.utexas.edu/users/tag/cmodels.html*
    DLV: *http://www.dbai.tuwien.ac.at/proj/dlv/*
    NoMoRe: *http://www.cs.uni-potsdam.de/~linke/nomore/*
    Smodels: *http://www.tcs.hut.fi/Software/smodels/*
    PSmodels: *http://www.tcs.hut.fi/Software/smodels/priority/*

Decentralized Web Service Organization Combining Semantic Web and Peer to Peer Computing

Shoujian Yu[1], Jianwei Liu[1], and Jiajin Le[2]

College of Information Science and Technology, Donghua University,
1882 West Yan'an Road, Shanghai, China, 200051
[1] {Jackyysj, Liujw}@mail.dhu.edu.cn
[2] Lejiajin@dhu.edu.cn

Abstract. While Web services already provide distributed operation execution, service publication and discovery with UDDI (Universal Description, Discovery and Integration) is still based on a centralized design. In this paper, we present an approach for distributed Web services organization by combining the capabilities of Semantic Web services with the dynamics and real-time search capabilities of peer to peer (P2P) networks. Furthermore, we use domain ontology to provide enhanced semantic Web service annotation. A DHT (distributed hash table) based catalog service is used to store the semantic indexes for direct and flexible service publication and discovery. For partnership federation, we have made improvements for this architecture. We use category ontology to organize domain specific service enabling semantic classification of service based on domains. Thus service publication and discovery is within the federate partners. We have proposed the decentralized Web service organization architecture and discussed the procedure of service publication and discovery.

1 Introduction

Web service paradigm has emerged as an important mechanism for interoperation amongst separately developed distributed applications in the dynamic e-business environment. An application can be built by integrating multiple services together making a more complex service. In typical Web service architecture, a public registry is used to store information about Web services description produced by the service providers and can be queried by the service requesters for specific application. However, the potential of a large scale growth of private and semi-private registries is creating the need for an infrastructure which can support discovery and publication over a group of autonomous registries. Emerging P2P solutions particularly suit for the increasingly decentralized application. They make it possible for different participants (organizations, individuals, or departments within an organization) to exchange information in a more flexible and scalable way. Recently a new generation of P2P

* This work has been partially supported by The National High Technology Research and Development Program of China (863 Program) under contract 2002AA4Z3430.

systems, offering distributed hash table functionality, have been proposed. These systems greatly improve the scalability and exact-match accuracy of P2P systems, but offer only the exact-match query facility. Their applications are extremely limited. Semantic Web technologies have been shown to support all these missing types of functionality [1]. Thus the combination of concepts provided by Semantic Web and P2P seems to be a good basis for the future of distributed Web service integration.

In our work, we use domain ontology for semantic Web service annotation. A DHT based catalog is used to store the semantic indexes for direct and flexible service publication and discovery. With a large number of electronic commerce application, each only focuses on publication or discovery of services with interest to their respective business domains. They are reluctant to partake irrelevant information of catalog services. Also, searching for a particular Web service would be very difficult and inefficient in an environment consisting of thousands of service providers from different business domain. In our work, we organize domains into category ontology for partnership federation. This ontology maps each service to a specific domain thereby grouping them based on domains. In this way Web services publication and discovery could be limited to only the services in that specific domain.

The rest of this paper is structured as follows. Section 2 briefly lists the related works. Section 3 presents an ontology based Web service annotation method. The detailed explanation for distributed Web service organization model is discussed. In section 4, we have proposed the improved Web service organization model, which supports domain specific service publication and discovery. Section 5 presents an initial implementation. Section 6 gives a conclusion and some future work.

2 Related Work

Currents approaches for Web service organization can be broadly classified as centralized or decentralized. The typical centralized approach includes UDDI [2], where central registry is used to store Web service descriptions. Three major operators, namely IBM, Microsoft, and ARIBA provide public UDDI service. The current UDDI attempts to alleviate the disadvantages of the centralized approach by replicating the entire information and putting them on different sites. Replication, however, may temporarily improve the performance if the number of UDDI users is limited. But, the more the replication sites, the less consistent the replicated data will be.

Having realized that replicating the UDDI data is not a scalable approach, several decentralized approaches have been proposed. [3] has proposed moving from a central design to a distributed approach by connecting private registries with P2P technology, but it does not consider partner federations in a specific domain. [4] organizes peers into a hypercube. But maintaining the hypercube with large amount of peers is inefficient. It does not support domain specific service federation either. [5] uses a dimension reducing indexing scheme to map the multidimensional information space to physical peers, which supports complex queries containing partial keywords and wildcards. But it does not utilize the semantics description. P2P based Web service discovery is also discussed in [6, 7]. None of these works consider domain specific service publication and discovery. Our work is different as we use category ontology to categorize Web service on the basis of business domains. We also use domain ontology to capture domain knowledge and index the ontology information, store the index at peers in the P2P system using a DHT approach.

3 DHTs Facilitated Decentralized Web Service Organization

Without a central services registry, a naïve way to discover a service in distributed system is to send the query to each of the participant, i.e. service provider. While this approach would work for a small number of service providers, it certainly does not scale to a large distributed system. Hence, when a system incorporates thousands of nodes, a facility is needed that allows the selection of the subset of nodes that will produce results, leaving out nodes that will definitely not produce results. Such a facility implies the deployment of catalog-like functionality.

Recently, a new generation of P2P systems offering DHT functionality has been proposed. These systems greatly improve the scalability and exact-match accuracy of P2P systems. This DHT functionality has proved to be a useful substrate for distributed systems. A DHT based catalog service can be used to determine which nodes should receive queries based on query content. Ideally, the P2P service network will allow for issuing a query to be sent to exactly those peers that can potentially answer the query.

In this section, we will discuss DHT based decentralized Web services organization model. The DHT background and Chord protocol are first introduced. Based on this, we propose the system model for semantics indexing catalog service. Then the service discovery procedure is illustrated. At the end of this section, we have discussed the maintenance of system evolution.

3.1 DHT Background and Chord

P2P is the decentralization from traditional central model to the decentralized service-to-service model. In this model no central index is required to span the network. Particular attention has been paid into making these systems scalable to large numbers of nodes, avoiding shortcomings of the early P2P pioneers such as file sharing systems like Gnutella [8] and Napster [9]. Representatives of scalable location and routing protocols are CAN [10], Pastry [11], Chord [12] and Tapestry [13], henceforth referred to as Distributed Hash Tables (DHTs). Given a key, the corresponding data item can be efficiently located using only $O(\log n)$ network messages where n is the total number of nodes in the system [12]. Moreover, the distributed system evolves gracefully and can scale to very large numbers of nodes. Our work leverages this functionality to provide a scalable fully distributed catalog service. [14] show that Chord's maintenance bandwidth to handle concurrent node arrivals and departures is near optimal. Thus we choose Chord as our DHT protocol.

Chord is a famous DHT protocol. Chord identifiers are structured in an identifier circle, which is called Chord ring. There is one basic operation in the Chord systems, *lookup(key)*, which returns the identity of the node storing the corresponding data item with that key. This operation allows nodes to publish and query information based on their keys. The keys are strings of digits of some length. Nodes have identifiers, taken from the same space as the keys (*i.e.*, same number of digits). Chord uses hashing to map both keys and node identifiers (such as IP address) onto the identifier ring (Fig. 1). Each key is assigned to its successor node, which is the nearest node traveling the ring clockwise. Depending on the application this node is responsible for associating the key with the corresponding data item. Thus it allows data request to be sent obliviously of where the corresponding items are stored. Nodes and keys may be added or removed at any time, while Chord maintains efficient lookups using just *O(logn)* state on each node in the system. For a detailed description of Chord and its algorithms, refer to [12]. Chord protocol is publicly available and has been successfully used in other projects such as CFS [15]. Nevertheless, our design does not depend on the specific DHT implementation and can work with any DHT protocols.

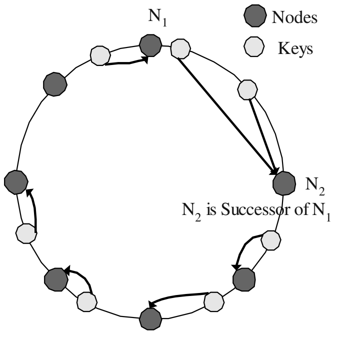


Fig. 1. The Chord identifier ring

3.2 Web Service Semantic Annotation Based on Domain Ontology

Adding semantics to Web service descriptions can be achieved by using ontologies that support shared vocabularies and domain models for use in the service description [16]. We refer to related concepts in ontologies as annotation. While searching for Web services, relevant domain specific ontologies can be referred to, thus enabling semantic matching of services.

An ontology is a shared formalization of a conceptualization of a domain [17]. In our approach, we identify two aspects of semantics for Web service description elicited from object-oriented model. We also model concept from both attributes and behaviors aspects. As shown in Fig. 2, the concept *Itinerary* has attributes as *startPlace, arrivalPlace* and *startTime* etc. It is also associated with behaviors as *search, book, cancel* etc. The concept describes what functionality a Web service aims at. The behaviors aspect can be used to describe operations of a Web service. The attributes aspects can be used to describe the i/o interface of operations.

WSDL (Web Services Description Language) is the current standard of the description of Web service [18]. The syntax of WSDL is defined in XML Schema.

From the left part of Fig. 2, we can see the simplified WSDL structure. We attempt to use minimum information to give Web service comprehensive annotations. Large distributed system can't afford much detailed information although ontology can capture much information of real world knowledge and domain theory. This kind of information can be incorporate into WSDL or UDDI. By this method, Web service is semantically annotated in different granularity, which facilitates different service level discovery (see Fig.2). The following code exemplifies Web service annotation method.

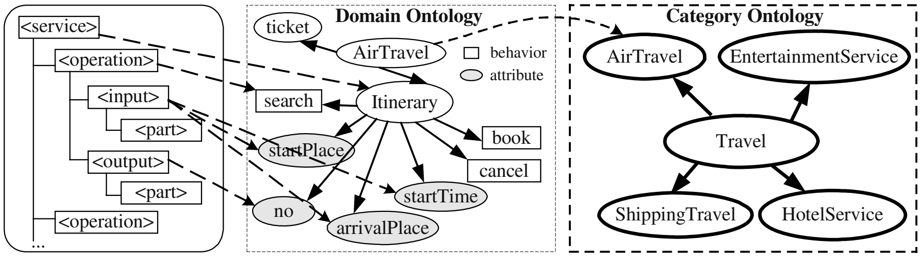


Fig. 2. The left part illustrates the simplified WSDL structure. The middle part is an ontology example. The arrow indicates semantic Web services annotation. The right part illustrates example of category ontology, which will be introduced in Sect. 4.1

```
<types>
  <xsd:element name="StartCity" OntCo-
cept="Itinerary:startPlace" minOccurs="1" maxOccurs="1"
type="xsd:string"
  <xsd:element name="ArrivalCity" OntCon-
cept="Itinerary:arrivalPlace" minOccurs="1" maxOc-
curs="1" type="xsd:string"
  <xsd:element name=" time" OntCon-
cept="Itinerary:startTime" minOccurs="1" maxOccurs="1"
type="xsd:string"
  ......
</types>......
<portType>
  <operation name="SearchItineary" OntCon-
cept="Itinerary:search">
  ......
  </operation>
</portType>  ......
<service name="ItineraryService" OntCo-
cept="AirTravel:Itinerary"
  ......
</service>
```

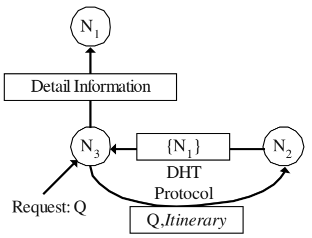
We add semantics to Web services by mapping service, operation, input and output in their descriptions to concepts in the domain specific ontologies. As the above WSDL code shows, the Web service is annotated in three aspects: service *Itinerary-*

Service is annotated by *AirTravel:Itinerary*, operation *SearchItineary* is annotated by *Itinerary:search* and i/o element *StartCity* is annotated by *Itinerary:startPlace* etc.

3.3 DHT Based Web Services Publication and Discovery

Let $\{N_i\}$ denote the n providers (a provider is a node in the identifier ring), each of which publishes a set C_i of Web services. When a node N_i wants to publish Web services, it creates catalog information, which is the set $C_i = \{(k_j, S_{ij}) \mid S_{ij}$ is a summary of k_j on node $N_i\}$. In Sect. 3.2, we have discussed the semantic annotation of Web services. Thus the concept should be the key, i.e. k_j, as mapped by Chord protocol and *Behavior(Attributes Set)* should be the corresponding data item associated with the key, i.e. S_{ij}, the summary of k_j. As a Web service consists of several operations, there are sets of data summaries S_{ij}, and not just single data summary.

For a service discovery, the requestor should first choose appropriate concept to which the preferred service may refer. Then he (or she) decides the interested behaviors and attributes he would like to provide as input for the preferred Web service. As an example, a service requestor wants to inquire about the itinerary information from Shanghai to Beijing on 1 October 2004. Considering the ontology in Fig. 2, the requestor may choose *Itinerary* as the domain concept, *search* as behavior and *startPlace, arrivalPlace, startTime* as attributes. This is the first step for service discovery.



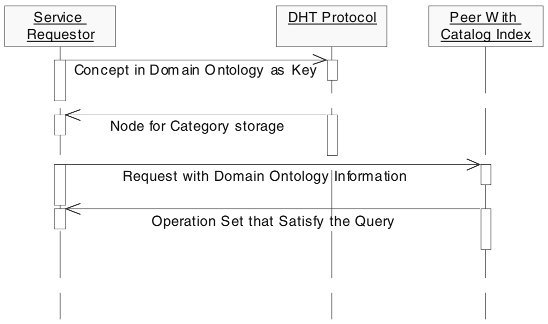

Service Requestor
DHT Protocol
Peer With Catalog Index
Concept in Domain Ontology as Key
Node for Category storage
Request with Domain Ontology Information
Operation Set that Satisfy the Query

Fig. 3. The left part illustrates Web Service Discovery Procedure. The right part illustrates interaction of DHT facilitated Web services discovery process

An example illustrates services discovery procedure. We assume that there are four services providers: N_1, N_2, N_3 and N_4. The service request is submitted on N_3. *Itinerary* serves as the DHT lookup key and the hash algorithm returns that the summary of *Itinerary* lies in N_2, which stores part of the catalog that contains *Itinerary* information. Then the request with domain ontology information is sent to N_2. This is the second step (see Fig. 3). On N_2, the behaviors and attributes of the query is matched against $S_{i,\text{ Itinerary}}$, $(1<i<3)$. N_2 replies to N_3 with the node set $\{N_1\}$. Here we assume that only $S_{1,\text{ Itinerary}}$ matches the given query and so N_1 is the only node that satisfies the request. This is the third step. Finally, N_3 contacts N_1 for detail information, e.g.

WSDL. Fig. 3 illustrates the procedure of Web services discovery and the interaction of DHT facilitated Web services discovery process.

3.4 System Evolution

When a node joins/leaves the system, the affected data structure on some existing nodes must be updated accordingly to reflect the change. This section describes how the distributed catalog service evolves when nodes join, leave and update their data, and how objects, which are the sets of data summaries, are stored and accessed.

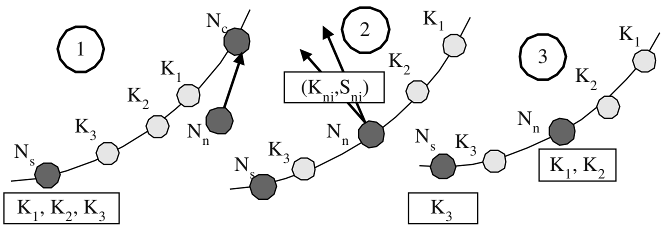


Fig. 4. When a Service provider joining the system, three steps is needed for system evolution

Nodes Joining

Each new node N_n (service provider) that joins the system creates the Service set C_n, which should be queryable by the nodes already in the system. First N_n contacts any node N_c in the system (Step 1, Fig. 4). Node and key have the same string space. N_n gets a position in the identifier ring by the same hash algorithm. Chord finds N_n's successor N_s in the identifier ring. The new node N_n, now part of the identifier ring, injects each $(k_{ni}, S_{ni}) \in C_n$ into the system and the Chord protocol decides which nodes should receive the new catalog information (Step 2). Additionally, N_n becomes part of the catalog infrastructure and should share the load of the catalog service by hosting parts of the summary sets already in the network. The Chord protocol will assign to N_n keys for which N_n should be the successor in the identifier ring. Therefore when N_n joins, k_1, k_2 and their corresponding summaries should be reallocated from N_s to N_n (Step 3), as shown in Fig. 4.

Updates and Departures

Catalog information stored in the system may need to be updated as the provider makes changes to their services. Update requests are handled in a similar way as insertion of information during nodes joining (Step 2, Fig. 4). Only the node that has created the data summary (the *owner*) is allowed to change its content. Nodes that store the data summaries are not allowed to alter their content, although they may alter the way they are stored. When a node N decides to leave the system, it must hand over the catalog information to its successor according to the Chord protocol. Furthermore, it notifies the nodes that hold N's catalog data. To achieve this, N uses the keys it inserted into the system to find the nodes that currently hold N's catalog information.

4 Category Ontology Facilitated Web Service Organization for Partnership Federation

DHT based Web service discovery realizes the distributed service discovery. It doesn't require a public service registry. But with the large number of services, every peer shares much catalog information irrelevant to its interest. The challenge of dealing with thousands of services during service publication and discovery becomes critical. In this section, we propose an improved architecture which allows peers form federate partnership with common domain interests.

4.1 Organizing Domain into Category Ontology

If we could categorize all these distributed services based on different business domains, finding the right services would be easier. In our approach, we create another kind of ontology: category ontology. The category ontology is used to categorize Web service based on domains and it maintains relationship between domains. It maps each service to a specific domain thereby grouping all the Web services based on domains. This allows that domain related services are assembled together. Thus finding a specific Web service in a specific domain could be limited to only those domain related services. As shown in Fig. 2, the right part presents an example of category ontology, and the middle part is the domain ontology.

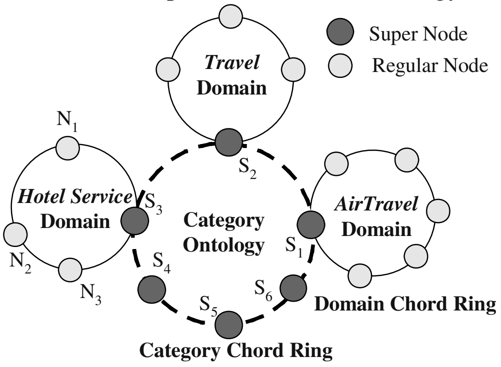


Fig. 5. Category ontology facilitated Web service organization for partnership federation

4.2 Improved Web Service Organization Architecture

The improved decentralized Web service organization architecture is shown in Fig. 5. There are two kinds of Chord ring for peer node organization. *Domain Chord ring* is used for organization of specific domain services. The *category Chord ring* links the separate domain specific services in their respective *domain Chord ring* together. The intention of this framework is to promise a pure P2P network suit for distributed application. The node in category Chord ring is called super node. Other nodes are called regular nodes. The *domain Chord ring* functions the same as in Sect. 3 for domain specific services publication and discovery. The *category Chord ring* differs from *domain Chord ring* in the information being routed: the *category Chord ring* provides category ontology catalog service for each *domain Chord ring*. That is to say, the *category Chord ring* routes category information to each super node in its Chord ring. Thus the super node shoulders not only the Web service information in its domain but also part of the category ontology information.

4.3 Web Services Publication and Discovery

When a service provider joins the infrastructure to publish his service, he has to choose an appropriate domain to which his business maps. A service provider is also allowed to map to multiple domains. If a new service provider wants to map to a domain that does not exist in the category ontology, he is allowed to create a new domain to map. Hence the service provider can either associate to an existing domain in the ontology or he can update the ontology with an appropriate domain and associate to that new domain.

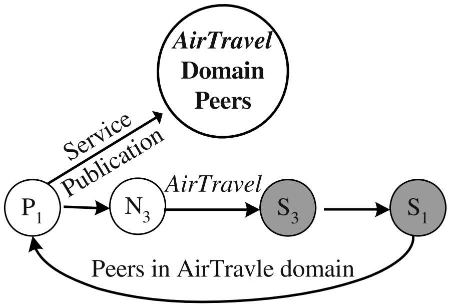


Fig. 6. Web service publication procedure combing domain ontology and category ontology

Considering the category ontology in Fig. 2, we assume that the DHT protocol routes *Travel* to node S_2, *AirTravel* to S_1 and *Hotel Service* to S_3, as shown in Fig.5. If a service provider P_1 in *AirTravel* domain wants to publish his service, we assume that he first contacts with node N_3 in *Hotel service* Domain Chord ring. N_3 contact the super node S_3 in its Chord ring and issues the *AirTravel* category information to S_3. Based on the DHT algorithm, S_3 knows that S_1 is the super node for *AirTravel* Domain. Then S_1 returns the peers in *AirTravel domain Chord ring* and P_1 contacts with any peer in *AirTravel* domain for service publication. The left service publication procedure acts as the same with in Sect. 3. Fig. 6 illustrates service publication process. Service discovery is similar to service publication. Service requestor should first refer to a domain in category ontology. The super node returns the domain specific peers and the requestor discovers appropriate services with the same procedure discussed in Sect. 3.3.

5 Implementation and Experiments

We have implemented an initial prototype to illustrate the organization model we have proposed in Sect. 3. The prototype is based on the Chord protocol implementation found on the Chord project website which is linked as a library. WordNet 2.0, an on-line lexical database for the English language, is embedded into the system through APIs to further understand the semantics of Web services described [19]. As the overlay network configuration and operations are based on Chord [12], its maintenance costs are of the same order as in Chord. An evaluation of the Web service discovery exactness and the system scalability is presented below.

5.1 Web Service Discovery Exactness Evaluation

To test service discovery exactness we first obtain a corpus of Web services from SALCentral.org and XMethods.com. We have limited our testing to two domains, i.e. Weather and Geographical domains, due to lack of relevant domain specific ontologies. We compare the discovery results in two circumstances: (a) Web service is not annotated with ontology information. The service names are directly used as the key hashed to catalog indexes. (b) The Web services are manually annotated with domain ontology. From the results in Table 1, we can conclude that method (a) has low correct rate and error rate. Comparatively, method (b) gets most of the services satisfying the request.

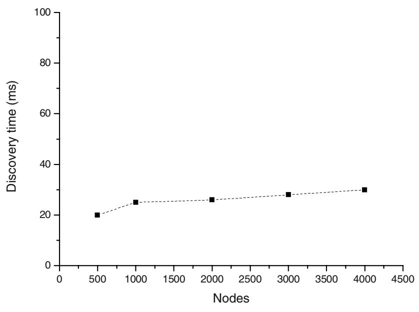
Table 1. Web service discovery exactness evaluation results: a collcetion of Web services in Weather and Geographical domains are used as samples

	Weather Domain		Geographical Domain	
	Correct Rate	Error Rate	Correct Rate	Error Rate
(a)	20%	0	17%	0
(b)	88%	0	91%	0

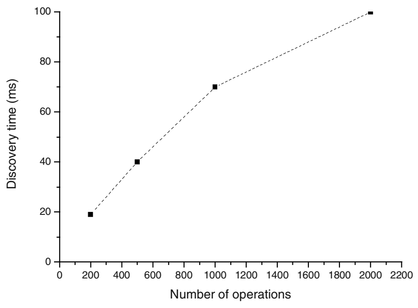
5.2 System Scalability Evaluation

In this section, we will evaluate the system scalability by constantly changing the number of nodes and operations in Web service discovery.

From (a) in Fig. 7, we can see that the discovery time changes steadily with peer nodes increase. This is because the discovery time mainly results from the matching of query against the summary in the catalog index. Thus it is not obviously irrelevant to the number of nodes. On the other hand, with the increasing of number of operations, the catalog index becomes larger, which results in longer discovery time as show in (b) of Fig. 7. Because the matching is processed only in one node, the increase ration of discovery time is not obvious. It inclines to stabilization. Thus this is not a critical problem for application in large distributed system. Also this can be overcome by improving the matching algorithm.



(a)



(b)

Fig. 7. System scalability evaluation results: we evaluate system scalability by evaluating the discovery time with number of nodes changes (a) and number of operations changes (b)

6 Conclusions and Future Work

This paper presents a flexible Web service organization architecture for service publication and discovery by combining semantic Web service with P2P networks. This system does not need a central registry for Web service discovery. We use an ontology-based approach to capture real world knowledge for semantic service annotation. A DHT based catalog service is used to store the semantic indexes for direct and flexible service publication and discovery. For partnership federation, we have made improvements for this architecture. We use category ontology to organize Web service into a specific business domain. Service publication and discovery is within a specific domain. Our initial experiments have shown that the semantic annotation approach suggested in this paper will significantly improve Web services discovery exactness. With Web services being as the enabling technology for next generation network, we believe that this service organization infrastructure will help organizations and businesses in carrying out their business goals in a more scalable environment.

Due to lack of relevant specific ontologies, we have not conducted experiments about category organization. In the future work, we will put this distributed Web service organization architecture on the Web, which will enable different organization and research workshop to test it. This will give more comments for us to improve this work.

References

1. R. Siebes: Peer-to-Peer solutions in the Semantic Web context: an overview. EU-IST Project IST-2001-34103 SWAP, Vrije Universiteit Amsterdam (2002)
2. UDDI. UDDI white papers. http://www.uddi.org/whitepapers.html

3. U. Thaden, W. Siberski, and W. Nejdl: A Semantic Web based Peer-to-Peer Service Registry Network. Technical Report, University of Hanover, Germany (2003)
4. M. Schlosser, M. Sintek, S. Decker and W. Nejdl: A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services. Second International Conference on Peer-to-Peer Computing (P2P'02), Linkoping Sweden (2002)
5. C. Schmidt, M. Parashar: A Peer to Peer Approach to Web Service Discovery. World Wide Web, Kluwer Academic Publishers, Vol. 7, 2 (2003) 211–229
6. M. Paolucci, K. Sycara, T. Nishimura, N. Srinivasan: Using DAML-S for P2P Discovery. Proceedings of the First International Conference on Web Services (ICWS'03), Las Vegas USA (2003) 203–207
7. A. Maedche, S. Staab: Services on the Move - Towards P2P-Enabled Semantic Web Services. Proceedings of the 10th International Conference on Information Technology and Travel & Tourism, Helsinki Finland (2003)
8. Gnutella Resources. http://gnutella.wego.com
9. Napster. http://www.napster.com
10. S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker: A Scalable Content-Addressable Network. Proceedings of ACM SIGCOMM2001, San Diego CA USA (2001)
11. A. Rowstron, P. Druschel: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms. Heidelberg Germany (2001) 329–350
12. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan: Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. IEEE/ACM Transactions on Networking. Vol. 11, 1 (2003) 17–32
13. B. Y. Zhao, J. Kubiatowicz, A. Joseph: Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. U. C. Berkeley Technical Report UCB/CSD-01-1141 (2001)
14. D. Liben-nowell, H. Balakrishnan, D. Karger: Observations on the Dynamic Evolution of Peer-to-Peer Networks. Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge MA (2002) 22–33
15. F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica: Wide-area cooperative storage with CFS. Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Canada (2001)
16. R.Akkiraju, R.Goodwin, P.Doshi, S.Roeder: A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. Proceedings of the Workshop on Information Integration on the Web (IIWeb'03), Acapulco Mexico (2003)
17. M. Uschold and M. Grüninger: Ontologies: Principles, Methods and Applications. Knowledge Engineering Review, Vol. 11, 2 (1996)
18. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. http://www.w3.org/TR/2004/WD-wsdl20-20040326/ (2004)
19. G. Miller: Special Issue, WordNet: An on-line lexical database. International Journal of Lexicography. Vol. 3, 4 (1990) 235–312

Modeling the Variability of Web Services from a Pattern Point of View

N. Yasemin Topaloglu[1] and Rafael Capilla[2]

[1] Department of Computer Engineering, Ege University, Izmir, Turkey
yasemin@bornova.ege.edu.tr

[2] Department of Informatics and Telematics,
Universidad Rey Juan Carlos, Madrid, Spain
rcapilla@escet.urjc.es

Abstract. In the last couple of years, Web services have been used in the development of Web systems, which are often composed by interoperable components. These interoperable components cooperate under a Client/Server style to accomplish the application integration and it is aimed to use the service components in several systems. Therefore, the development of such Web services, as high quality and reusable components is a goal to achieve. In this sense, suitable design and modeling techniques can improve the customisation of similar services built by one or more service provider. On the other hand, the development, the management and evolution of many modern software systems rely on the notion of variability and pattern-based approaches as suitable design techniques. In this work we will try to describe how the development of Web systems using Web services can be improved with a suitable description of the variability as a technique for building and customizing similar systems. We will focus on the modelling of variability of Web services from a pattern point of view.

1 Introduction

In the last decade the Web has become the user interface for many information systems that are distributed across Internet or corporate Intranets. Due to this, the development of many information systems has migrated to include a Web interface or they have been reengineered as Web-based systems. Also, the demands placed on Web-based systems and the complexity for designing, developing, maintaining and managing such systems has increased significantly [12]. Moreover, due to the recent trends in B2B applications, the integration of Web applications employing different components and technologies becomes a high-priority goal in the development of Web information systems.

A Web service represents a software component, which is registered, published and called under standard Web protocols. Web services [1] enable the development of interoperable Internet applications that realize the idea of *software as a service* [20]. Companies that require application interoperability such as those in the financial services, energy trading, high-tech manufacturing, tourism reservation systems and telecommunications industries are ideal candidates for Web services technologies [7].

However, the development and management of such components require suitable development techniques to realize the potential benefits. Within this respect, the concept of variability and variability modeling has a considerable importance, since Web services are intended to be used in several systems through customization.

In this work we focus on architectural and pattern approaches for describing the variable points (i.e.: variation points) of Web services in order to improve the design process. In section 2 we describe which representation techniques can be used to describe the variability of systems. In section 3 we relate the notion of variability with the architectural representation of Web services from a pattern point of view. Section 4 discusses the internal representation of a Web service through the description of suitable variation points for customizing and integrating such services more easily. Finally, in section 5 we provide the conclusions of our work.

2 Variability Representation Techniques

The notion of variability relies on the concept of variation point (VP) [3] and variants to discriminate different features of a system. The definition, representation and categorization of suitable variation points can provide different versions and configurations of systems derived from the same architecture [2]. This customization process is performed through specific variation points that are personalized at several stages (i.e.: the binding time) in the development process. The need to describe the variability of systems has lead to different proposals from different authors but there is no agreement about which is the best mechanism or representation technique for representing the variation points at the design level.

In this way, the FODA model proposes a featured model with mandatory, alternative and optional features to describe different characteristics of systems. Other proposals try to improve this model by adding quantitative values to variation points using a simple language [5]. Also in [17], the authors mention the existence of several graphical notations for representing the variability issues from the same point of view (i.e.: a featured view). FODA has also been used to model web services variability [18]. For non object-oriented systems such as most common Web applications, the authors [6] propose an independent representation form to describe variation points at the pattern or architectural level. A negative aspect is that most of the proposals mentioned lack of a standard notation and exchange format, such as mentioned in [10], for representing the variability issues, while other proposals use UML. The main advantage of using UML is that is a standard notation accepted by the software engineering community. In the opposite side we can say that the expressiveness of the language could not be enough for representing all the aspects needed. Therefore, UML stereotypes or other mechanisms are used for this purpose. Part of the information of the variation points (e.g.: rationale, variants, multiplicity or binding time) are included in UML diagrams in order to formalize the variability issues [15] many times described with textual information or other kind of diagrams. In other cases the information shown by UML diagrams doesn't represent with enough detail the information defined in the variation points.

But when describing the design of a system, most of the boxes representing modules or subsystems don't specify which of them hold variation points that later will be mapped to specific software components. This must be done explicitly if we want to outline which parts of the architecture represent variations in the future system. The description of these variation points can be done with more or less detail but a more accurate description of these variation points, their variants (e.g.: attributes), values and ranges permitted should be listed, stored and managed properly (e.g.: using databases or specific tools).

In next sections we will relate the modeling task of Web services using the notion of variability described before. From our point of view, we have divided the design of Web services from the following two perspectives: an *architecture-oriented* to describe those aspects related to protocols, environments, distribution, and a *task-oriented* one to describe the internals of a Web service that is the functionality performed by the service.

3 Architectural Modeling Aspects of Web Services

Web services are a key piece in the development of many modern software systems because they enable the decomposition of a system into modules cooperating among them under a Client/Server style and because they employ reusable software components that can be discovered and called under Internet protocols. The interoperability between applications and services is usually provided by standards based on XML such as SOAP, WSDL and UDDI [9]. Web services can be seen as an outcome of multi-tier applications that separate the business logic and the presentation layers. In this way, the provider of a service presents the interface of the service to their clients. Therefore, we can use the *Broker* architectural pattern and the *Client-Dispatcher-Server* design pattern [4] for modeling purposes. For instance, the Broker pattern can be seen as a module for finding or connecting different services across the Internet, such as a flight reservation application where a Broker finds the complete information provided by several services and show this to the user that books a flight in a transparent way. The *Client-Dispatcher-Server* pattern introduces an intermediate layer between clients and servers (i.e.: the *dispatcher* component). The dispatcher provides location transparency by means of a name service, and hides the details of the establishment of the communication between clients and servers [4]. Figure 1 shows the *Client-Dispatcher-Server* design pattern.

In the schema proposed in figure 1, the role of the dispatcher could be associated to a UDDI repository for registering and discovering services. A naming service will connect clients and servers transparently without worrying the clients about the specific location of a service.

The standards used in Web services technologies provide several flexibilities from an architectural point, which can be seen as variability issues, such as the following ones:

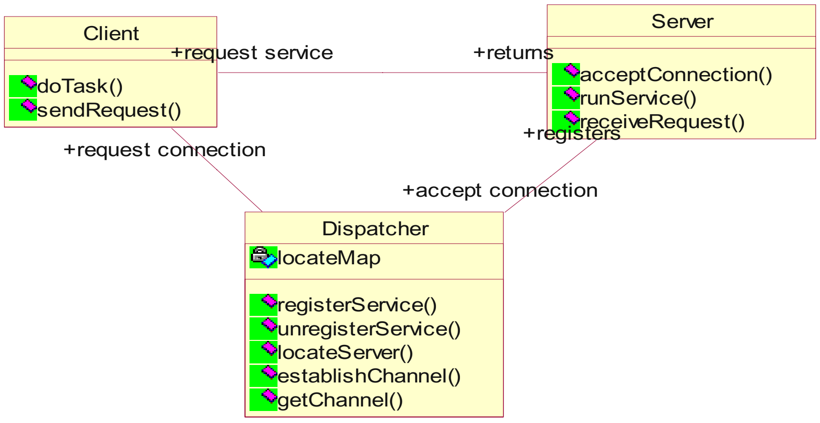


Fig. 1. The Client-Dispatcher-Server Design Pattern.

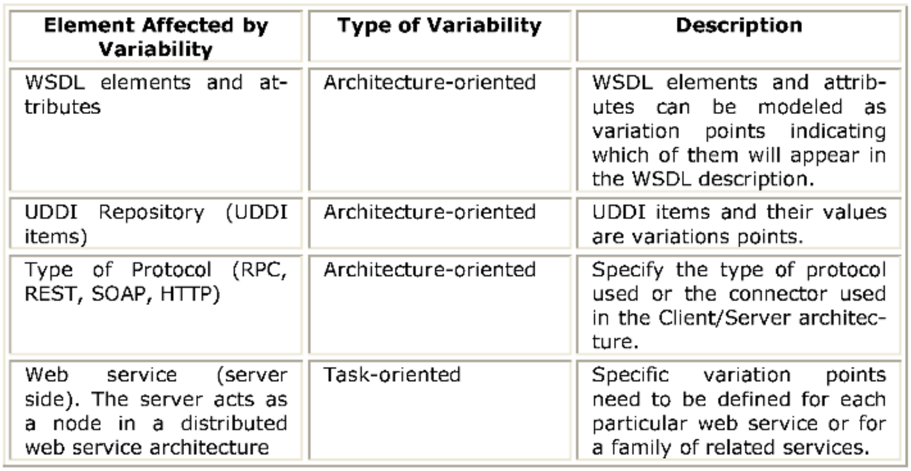
(a) Client and servers can be members of different environments. For example, a Java application can invoke a Web service written in ASP.NET or a Web service can be used to communicate with a database from a legacy application.

(b) Different protocols can be used to transfer a SOAP message between the client and the server.

From the protocol point of view, the most common implementations of Web services employ a RPC (Remote Procedure Call) connector in the Client/Server style. The data elements are usually codified as XML documents and transmitted using the SOAP protocol as SOAP messages. Other similar approach is the REST style (Representational State Transfer), based also on the Client/Server model but applying constraints to the architectural elements. REST architectures allow resources that can be labeled and a REST application can transfer the state of the resource using a connector (i.e.: so-called a cache connector). The scalability, performance and other non functional properties employing RPC or REST [14] can be different when using Web services.

On the other hand, the server side must provide a suitable description of the Web service using WSDL and registering this under UDDI repositories for a subsequent discovering and retrieval process by the clients. In the case of WSDL, we can specify data and functions defined in a Web service as XML documents. A WSDL document defines services as collections of networks endpoints called *ports*. The elements defined in a WSDL document are: Types, Ports, Operation, Port Type, Binding, Port and Service and they include attributes (e.g.: name, element, type, etc.) that are filled before the message is sent. Such elements and attributes can be seen as customizable variation points, which can be represented under an architectural point of view. In the same way, the UDDI protocol for registering and discovering Web services comprises four core data structure types, the businessEntity, the businessService, the bindingTemplate and the tModel. Each of these four types contains several descriptive items about a business (e.g.: name, description, contacts, etc.), a logical service, a description of the Web services provided or entities for determining the compatibility of Web services. Table 1 summarizes the elements used in a Web service schema for which the variability can be defined and represented.

Table 1. Web services architectural elements candidate to be described by suitable variation points.



Element Affected by Variability	Type of Variability	Description
WSDL elements and attributes	Architecture-oriented	WSDL elements and attributes can be modeled as variation points indicating which of them will appear in the WSDL description.
UDDI Repository (UDDI items)	Architecture-oriented	UDDI items and their values are variations points.
Type of Protocol (RPC, REST, SOAP, HTTP)	Architecture-oriented	Specify the type of protocol used or the connector used in the Client/Server architecture.
Web service (server side). The server acts as a node in a distributed web service architecture	Task-oriented	Specific variation points need to be defined for each particular web service or for a family of related services.

In addition to this, other variability options refer to the categorization of the services according to the functionality they expose as *Programmatic Web Services* (PWS) and *Interactive Web Services* (IWS) [21]. PWS expose function calls that correspond to the business logic tier of an application. A client of the Web service can build new applications by combining the Programmatic Web Services. On the other hand, Interactive Web Services expose an application's user interface layer and usually are created by using several technologies like a Web server, an application server and storage systems. IWS clients can incorporate those interactive business processes into their Web applications.

The architectural and the functional flexibilities of Web services enable the implementation of flexible software sytems which can be adapted according to the requirements. This flexibility can be modeled through the concept of variation point, seen as a key technique for building similar systems [13]. In order to make easy the design, modeling and customization of Web services, the use of specific variation points in the modeling task can facilitate the design and integration processes to improve the development of systems based on Web services as well as those aspects related to coordination and collaboration processes. Therefore, we need to define and represent the variability that affects to these services, such as we describe in next section.

4 Web Services Variability Representation

The functionality offered by a Web service is usually built as a set of functions that are transparently called by a software application via Web protocols. Such functionality can be implemented in several programming languages such as Java or C#. When a service is called, parameters can be passed and the result is then returned to the Web application. Therefore, an initial level of variability can be defined with respect to the functions offered by the service as well as for the parameters the Web services admit. Figure 2 shows an example of how the variation points can be modeled for a Web services using UML classes as an attempt to include a more accurate information compared to the description given in [18].

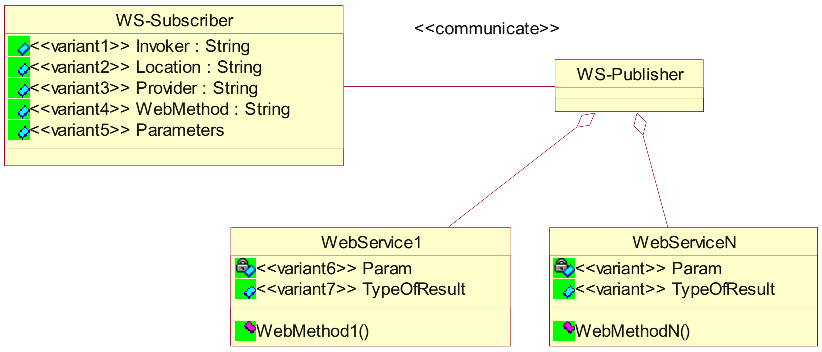


Fig. 2. The variability of a Web service is described for a subscriber and a publisher.

The left side of figure 2 defines a generic software application that calls a particular Web service. Each service can be located in the same place or in different locations (i.e.: different servers). The variability exposed in the client side represents the following information: the invoker of the service, the location, the provider, the name of the function called (i.e.: a particular service) and the parameters passed if needed. In the server side, the variability is defined for the functions or specific services offered. The parameters that characterize each function and the type of result returned by the service (e.g.: numeric, string, HTML, etc.) are the variation points defined. For instance, a client of a Web application for buying flight tickets calls a specific Web service for that purpose. The service of the provider seeks flights for a specific destination and dates and shows to the client an HTML page with the available flights and prices. The provider of the example has only one Web service with a unique method or operation but in most of the cases several services and a variety of methods will be offered by the service provider. The representation of the variation points following the schema given in figure 2 is shown in figure 3.

To represent the type of variation points mentioned above, design patterns [11], which represent a generic solution to be used in different contexts, could also be used.

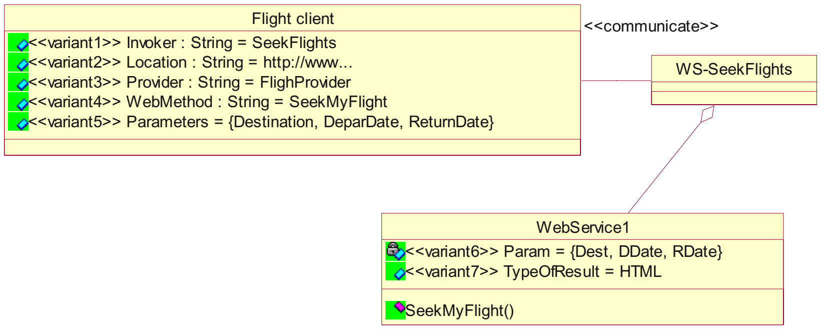

Flight client
<<variant1>> Invoker : String = SeekFlights
<<variant2>> Location : String = http://www...
<<variant3>> Provider : String = FlighProvider
<<variant4>> WebMethod : String = SeekMyFlight
<<variant5>> Parameters = {Destination, DeparDate, ReturnDate}
<<communicate>>
WS-SeekFlights
WebService1
<<variant6>> Param = {Dest, DDate, RDate}
<<variant7>> TypeOfResult = HTML
SeekMyFlight()

Fig. 3. Example of variability modeling for a flight reservation Web service.

We will refer to a Statistical Analysis System that has been implemented with Java Web Services technologies to discuss variability modeling with design patterns. The motivation for the statistical analysis system is to allow researchers from different locations to access a powerful analysis implementer server to perform complex analysis tasks [19]. The system offers to researchers several statistical analyses services like: *completely randomized design, randomized block design, latin square design* and *regression analysis* which correspond to various analyses requests of different disciplines. It was aimed to provide flexibility both for the clients of the system and for the server; so several design patterns are employed in design and construction of the system. The design patterns enable to structure the identified variation points and also facilitate the maintenance tasks of the system.

Some variation points of the system and the design patterns used to represent these variation points are described below:

1. *Variation in the web service function* (Different analyses algorithms): The *Strategy* [11] design pattern that enables to define and also to encapsulate a family of algorithms is used to represent the various analyses algorithms used in the services. Therefore, the analyses algorithms can vary independently from the clients that use them. This can be seen as a very high level type of variability, not as concrete variation points that can be customized in a given moment of the development process.
2. *Variation in the required parameters* (Different type of data sent to analyses): The *Decorator* [11] design pattern that enables the allocation of responsibilities to an object's method dynamically is used to work with different types of data sent to the analyses. For example, the data sent to an analysis can be a single variable, an array of variables or a matrix of variables. With the help of the Decorator pattern, it becomes possible to add tasks dynamically to methods that deal with different types of data.
3. *Variation in the protocols* (Different protocols that clients may use to communicate with the analyses services): This is an example of an architectural variability, mentioned in Table 1. The *Adapter* [11] design pattern is used to

provide various interfaces that clients may require in the communication with the analysis service. The Adapter pattern enables to adapt the standard interface of an analysis service to communicate with clients that may use different protocols. Therefore the decoupling of the services implementations from the specific interface requirements of the clients becomes possible.

The examples of variability mentioned in the flight seeking example and the statistical analysis system mainly focus on the variability of Interactive Web Services defined in section 3. In next sub section we will mention to variability with respect to Programmatic Web Services.

4.1 Variability for Coordinating Web Services

Today, the development of many Web information systems trend to service oriented architectures, in which a set of services collaborate, transparently to the user, to produce some result. In this way, several Internet protocols such as BPEL4WS, BPML and WSCI have been proposed. This collaboration process can be established over the concepts of *choreography* and *orchestration* [16]. By choreography we mean a collaborative process that allows the interaction among different Web services working together, whereas the term orchestration refers to a business process that interacts with both internal and external Web services and controlled from one party's perspective. This result is often business driven, but there is a need to compose a set of business processes in an ordered way, such as the Programmatic Web Services (PWS) mentioned in section 3. A simple example of a system that relies on PWS would be a system that combines several services which are mathematical components in order to build a component with more complex computing capabilities.

Related to this, it is possible to describe some variable aspects when designing the composition of Web services from a dynamic point of view (e.g.: a sequence diagram) rather than from a static view. The variation points will affect to the description of the flow needed to perform a set of ordered operations, run-time conditions and responses executed by a particular business process. For instance, an e-commerce software application could implement a shopping card as a service. The functionality included in this service can be divided into two or more services that are called in an ordered way to achieve the proposed goal (e.g.: one Web service will seek the products while other one will add them to the shopping card).

From our point of view, the basic information for composing a service would be the number of sub-services that compose a service placed in a higher abstraction level and the order in which the services or functions are called to offer the functionality needed by the client. Other information can be also modeled as variation points in the server side such as service location, intermediate responses or error handling, but for an initial modeling task we consider this information would be enough. For instance, the flight reservation service shown in figure 3 can be composed by several sub-services that cooperate in a coordinated way to sell a flight to a customer. Usually, the reservation and purchase of the ticket is done in several ordered steps.

First, the customer seeks for flights for a specific date. Later the system provides a list of flights available for that date or a message if no flight appears. After, the customer can select different hours for flying and in some cases the system is able to seek a list of best prices from several flying companies. Finally, the customer buy the e-ticket filling the credit card and personal information and the payment is performed. Some of these tasks can be implemented by one or more services that are composed and called in an ordered way.

For instance, we can think that this process is divided in two sub-services, one for the flight seeking process and another one for the electronic payment. Then we can model the variability mentioned before for the server side. In figure 4 we show the new variation points added to figure 2 in order to model the composition and coordination aspects in the server side.

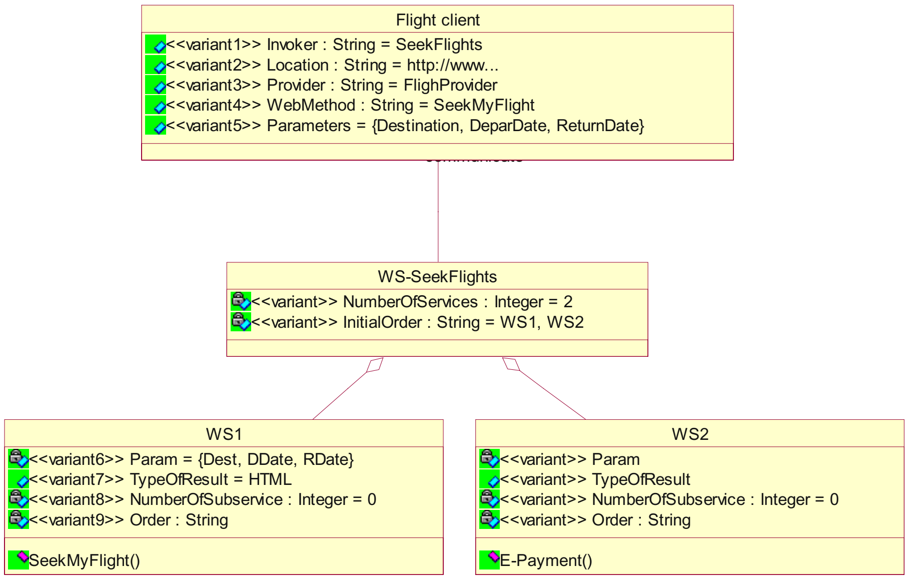

Flight client
<<variant1>> Invoker : String = SeekFlights
<<variant2>> Location : String = http://www...
<<variant3>> Provider : String = FlighProvider
<<variant4>> WebMethod : String = SeekMyFlight
<<variant5>> Parameters = {Destination, DeparDate, ReturnDate}
WS-SeekFlights
<<variant>> NumberOfServices : Integer = 2
<<variant>> InitialOrder : String = WS1, WS2
WS1
<<variant6>> Param = {Dest, DDate, RDate}
<<variant7>> TypeOfResult = HTML
<<variant8>> NumberOfSubservice : Integer = 0
<<variant9>> Order : String
SeekMyFlight()
WS2
<<variant>> Param
<<variant>> TypeOfResult
<<variant>> NumberOfSubservice : Integer = 0
<<variant>> Order : String
E-Payment()

Fig. 4. Variation points defined for composing and coordinating Web services for a flight reservation system.

To represent the composition of several Web services, the *Composite* [11] design pattern, which composes objects into tree structures to represent part-whole hierarchies seems suitable. The Composite design pattern would allow clients to treat individual Web services and compositions of Web services in the same way, meaning that the clients use a class interface, in the above example WS-Seek Flights, to request services from the Web services in the composite structure. If the recipient Web service of the request is a leaf, in the above example WS1 or WS2, the client's request is answered directly. If the recipient Web service is a composite Web service, like WS-Seek Flights, the client's request will be transferred after/before conducting additional tasks. In addition, the *Iterator* and *Chain of Responsibility* [11] design patterns, which usually accompany the Composite pattern, may be useful to model the coordination of Web services. The Iterator pattern enables to access the elements of an aggregate object sequentially without the need to know its underlying representation. Therefore, Iterator pattern may be employed in traversing the Web services sequentially, like WS1 and WS2 of WS-Seek Flights. On the other hand, the Chain of Responsibility pattern provides an alternative to Web Services by replacing each other in case of not being able to meet a specific request. For example, a client's request about payment, maybe handled by another Web service if the E-Payment results unsuccessfully.

5 Conclusions

The construction of many modern distributed information systems based on Web standards employ Web services as a hot new technology for improving the development and maintenance operations. The flexibility provided by service oriented architectures makes easy the transition from other rigid schemas because they distribute the development of software among different providers.

In this work we have used the notion of patterns to describe how Web services based systems can be designed including the notion of variability. In this way, the design of Web information systems employing the notion of software architectures and design patterns can be improved with a good description of the variation points that introduce flexibility in the design process. Moreover, the concept of variability facilitates the customization process when developing similar systems and this is achieved through the definition of suitable variation points. In addition to this, the use of product line approaches [8] to the development of Web services employing variation points could be quite useful for accelerating and improving the construction of reusable Web services as well as complex web sites.

Also, we have defined appropriate variation points using UML diagrams in order to facilitate the implementation and customization issues in the development process. Also, the definition of variation points for modeling the coordination and composition aspects facilitates the design of these tasks as well as the implementation issues. The applicability employing Web services is immediate because the modeling aspects and the customization of similar services built by one or more service provider can be improved. We advocate for new patterns to describe service oriented architectures that include variable aspects for customization and development purposes.

Finally, our discussion after figure 4 is an initial attempt to model Web services composition and coordination with known design patterns. New patterns that meet the unique requirements of Service oriented software architectures seems to be a hot topic for a near future.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Concepts, Architectures and Applications. Springer-Verlag Berlin Heidelberg (2004)

2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, Addison-Wesley, 2nd Edition (2003)
3. Bosch, J., Design & Use of Software Architectures, Addison -Wesley (2000)
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns, John Wiley & Sons, New York (1996)
5. Capilla, R., Dueñas, J. C.: Modelling Variability with Features in Distributed Architectures. 4th International Workshop on Software Product-Family Engineering. Lecture Notes in Computer Science, Vol. 2290. Springer-Verlag, Berlin Heidelberg New York (2002) 319–329
6. Capilla, R., Topaloglu N. Y.: Representing Variability Issues in Web Applications: A Pattern Approach. ISCIS'03, Lecture Notes in Computer Science, Vol. 2869. Springer-Verlag, Berlin Heidelberg New York (2002) 1035–1042
7. Chung, J., Lin, K., Mathieu, R. G.: Web Services Computing: Advancing Software Interoperability, IEEE Computer, October (2003) 35-37
8. Clements, P. Northrop, L. Software Product Lines. Practices and Patterns, Addison-Wesley (2002)
9. Curbera et al: Unraveling the Web Services Web, IEEE Internet Computing, March-April (2002) 86-93.
10. Dobrica, L., Niemelä, E.: Using UML Notation to Model Variability in Product Line Architectures, International Workshop on Software Variability Management, ICSE'03, Portland, Oregon, USA, (2003) 8-13
11. Gamma,E.,Helm,R.,Johnson,R.,Vlissides,J.: Design Patterns, Addison-Wesley (1995)
12. Ginige, A., Marugesan, S.: Web Engineering: An Introduction. IEEE Multimedia, January/March (2001) 14-17
13. Jacobson, I., Griss, M., Johnsson, P.: Software Reuse. Architecture, Process and Organization for Business Success, ACM Press (1997)
14. Mitchell, K.: A Matter of Style: Web Services Architectural Patterns, XML Conference & Exposition, http://www.idealliance.org/papers/xml02/dx_xml02/papers/03-02-03/03-02-03.pdf (2002).
15. Myllymäki, T. "Variability Management in Software Product Lines", Tampere University of Technology, Software Systems Laboratory, ARCHIMEDES, http://practise.cs.tut.fi/pub/papers/VarMgnFinal.pdf, 2001
16. Peltz, C.: Web Services Orchestration and Choreography, IEEE Computer, October (2003) 46-52
17. Robak, S.: Feature Modeling Notations for System Families, International Workshop on Software Variability Management, ICSE'03, Oregon, USA, (2003) 58-62
18. Robak, S., Franczyk, B.: Modeling Web Services Variability with Feature Diagrams, International Workshop on Web Services, Research, Standardization and Deployment, WS-RSD'02, Erfurt, Germany, (2002)
19. Topaloglu, N. Y., Atıl, H., Elmas T., Goknil, A.: A Statistical Analysis System with Java Web Services, International Congress on Information Technology in Agriculture, Food and Environment, ITAFE'03, Izmir, Turkey, (2003) 477-482.
20. Turner, M., Budgen, D., Brereton, P.: Turning Software into a Service, IEEE Computer, October (2003) 38-44
21. Web Services Architecture White Paper, http://www.webcollage.com/

Managers Don't Code: Making Web Services Middleware Applicable for End-Users

Alexander Hilliger von Thile, Ingo Melzer, and Hans-Peter Steiert

DaimlerChrysler AG – Research and Technology,
P.O. Box 2360, 89013 Ulm, Germany
{alexander.hilliger_von_thile, ingo.melzer, hans-peter.steiert}
@daimlerchrysler.com

Abstract. Today's web-pages are primarily designed for occasional usage. Professional users therefore use special applications that use Web Services increasingly. As the number of internet-users grows we argue that there is a disregarded growing gap between these professional- and occasional-users we refer to as experienced users. For this group of users with little or no programming-skills web-pages are inefficient but professional applications would be inexpedient. In this paper we describe how to make Web Services applicable for experienced web users. To support single Web Service calls efficiently we present a keyboard-controlled browser-embedded console with command auto-completion that wraps Web Services. To support multiple calls and automation we present a web-based IDE that allows visual composition of Web Service calls and simple control-structures that can be used on demand without installation and programming-skills.

1 Introduction

Since its birth, the internet has grown into a tremendous popularity. Today the number of users still grows and what's even more interesting: these users use the internet more frequently. As the internet was primarily used by programmers and scientist in its early years, it became a mainstream medium in the meantime. But most popular web-sites are primarily designed for occasional usage only. That's why professional users need to use special client-applications. The interface between these services and the application was based on HTTP-site requests and -responses in many cases where an official API was not available. The client-applications were sending HTTP-GET requests by automatically building URLs or HTTP-POST request by simulating a HTML-form transmission to the server. The response from the server – usually a HTML-page – was processed by special parsers. Several publications can be found about this way of web-based query and automation ([1], [2] and [3]). However, using web-pages as an interface has many problems mainly due to page-layout-dependencies [3]. Meanwhile many popular web-sites like Amazon.com and eBay provide official APIs using Web Services to ease program development for their services. But Web Services have programmers as their target group and are therefore uninteresting for the vast majority of internet-users. Today this group of users has only two choices: using the web-pages, which is inefficient for frequent, experienced users or buying a special application for every service, assuming that their usage-scenario is covered by an application at all. In this paper we describe how to fill this gap-in-between by providing mechanisms to make Web Services available to non-programmers.

We therefore focus on the group of experienced web-users that can be characterized as follows:

- they are non-programmers (little or even no programming-skills)
- use services on the web frequently
- web-site navigation is inefficient for their usage scenario (compare prices of multiple auctions, monitor weather-data, traffic-conditions or validate addresses)

If we take eBay as an example and assume that only 10% of eBay's customers fall into this category of users we would address the need of 9.49 million people [4].

The experienced web-users usage-scenarios can be divided into two categories:

- **Query:** (nested) Web Service calls are used to query specific data (i.e. prices)
- **Automation:** conditional expressions can be used to execute actions, such as a set of queried values can be compared and a specified action (nested Web Service calls) could be triggered (get cheapest price of all previously selected auctions $a_1..a_n$ and place bid if price is below my limit L)

In chapter 2 we describe mechanisms required to make Web Services accessible for experienced web-users. In chapter 3 we show how to support efficient queries and in chapter 4 how automation of queries can be realized. Section 5 describes related work. We conclude in chapter 6.

2 Considerations

Using Web Services directly without a toolkit by i.e. typing a SOAP-message is of course inefficient. In this section we want to analyze which requirements an appropriate toolkit for Web Service usage has to meet.

We identified the following requirements:

1. Abstraction from Web Service complexity (SOAP, HTTP-connection, bindings, endpoints)
2. Abstraction from programming-complexity (classes, functions, control-structures)
3. No local installations necessary (runtime environments, IDEs)

The first level of abstraction is realized by using an existing framework to wrap Web Services complexity. We used Apache AXIS [5] to generate stubs as Java-classes. We use Java because it is widespread, supports late binding that allows us to dynamically add classes during runtime and it has reflection-mechanisms allowing compiled classes to be analyzed for method- and field declarations. Java might be simple compared to other programming-languages but is still too complex for our target group. We therefore add another layer of abstraction by using JavaScript to use the generated Java-classes.

```
simple_statement ::= [objectName =]
                     (object_instantiation | object_method_call);

object_instantiation ::= new [package.]classname([parameterlist])

object_method_call ::= [objectName.]method_name([parameterlist])
                       {.method_name([parameterlist])}

parameterlist ::= (objectName | object_instantiation |
                  object_method_call)[, parameterlist]
```

Fig. 1. EBNF-notation of a simple JavaScript statement

JavaScript (later standardized as ECMAScript) [6] is an object-oriented programming language with an easy to learn syntax. For now let us focus on simple statements as defined in fig. 1 using EBNF notation [7]. In the Java-philosophy everything is an object. A program consisting of simple statements creates objects explicitly using the object_instantiation-statement (see fig. 1) or implicitly by using a number or defining a String using quotation marks. A name can be assigned to these objects optionally. This name is used to reference the object and call its member-functions (object_method_call) with parameters (parameterlist) which are also object-references (existing or newly created) or results of nested method calls. The return value – if present – can be assigned a name to as well.

Programs as described above are basically simple from a developer's perspective; however the JavaScript-syntax with its nested curly braces, brackets and parenthesis is still not suitable for non-programmers. In the next chapter we will therefore determine how to map above simple statement to a web-based query console meeting our abstraction requirements. To allow automation using control-structures (*if*, *for*, etc.), define event-listeners and allow scheduling of function-calls easily without the need to write code we will use a graphical model presented in section 4.

Requiring a user to read a manual to understand what methods a Web Service offers and how they are used is unacceptable for our target group. But modern IDEs features like code-completion (context-based list of variables and methods), method and parameter descriptions (such as JavaDoc of current method) require appropriate meta-data. The easiest way to get this meta-data is using a reflection mechanism that allows the extraction of methods with their parameter types from compiled classes. But neither the names of the parameters nor the usage-description can be reflected. However the WSDL-description contains the names of the parameters, since they are easy to parse the names can be extracted from them. Usage-description is not contained within the WSDL-file and even if a UDDI entry exists for the Web Service the usage-description is not machine-readable.

A lot of modern tools and programming languages support in place documentation. Two examples are JavaDoc and POD (plain old documentation, used by Perl). The idea is to maintain technical representation and the verbose documentation in one document. Many modeling tools also allow adding comment to diagrams and WSDL has the "documentation" tag which can be embedded at various places inside of a

WSDL-document. These pieces of information which are not very useful for computers reading the documents should be kept when generating code. For example, during the generation of Java stubs out of a WSDL-document, the verbal content of the source file should be transferred into JavaDoc.

At the moment, a useful execution of this step is not possible, because there are no specifications which clearly state how this additional documentation should be structured and which kind of information should be placed at which level. Therefore, if a generated document is used in a chain to generate a third document, an initially helpful verbal documentation will be misplaced and mostly useless.

Keeping documentation in separate files and using an external system for documentation might be a good additive, but is not enough to be used as the only documentation source. The risk of an old and outdated documentation is just too high. Developers will modify some files such as the code or the model and will "forget" to update the external documentation. Also, having the needed documentation right at your fingertips eases life a lot and reduces the risk of misinterpretations.

To meet the last requirement (avoid local installations), both solution strategies are web-based and controlled using a browser. Apart from the benefit that end-users do not have to cope with installation and administration procedures this solution enables platform-independent on-demand usage as well as user's scheduled program-execution even if their local workstation is offline.

3 Query-Console

Many people associate writing queries with writing SQL statements used to query database management systems. But most web-sites do not allow a direct access to their internal databases; their content is queried using predefined HTML-forms which limit the type of queries to a predefined subset defined by the site-owner [2]. Query-results are wrapped into HTML-pages making it difficult to extract them (see section 1).

Today more and more web-sites offer Web Services for automation and query. This reduces the problems of parsing semi-structured web-sites to get their relevant content but it does not solve the problem of how to query the content. The revealed APIs differ from service to service which means that using a new service to execute a query requires understanding the web-site's API first. Compared to the database-world this would mean that you not only have to understand the schema of a database but also have to understand the operations defined to access the data itself.

Web Service based queries are therefore characterized as programs with special hierarchical access operations defined by the service. Because they are programs they are in contrast to SQL non-descriptive and data-access is hierarchically navigation based like in hierarchical databases instead of (object-) relational because a predefined path (axis) exists to the data.

In this section we want to analyze how to enable non-programmers to efficiently execute queries based on Web Service-calls. To avoid the full complexity of programming we will confine on the functionality described in section 2 that can be realized as follows.

F1: **Prepare a Web Service for first time usage**: this requires specification of the wsdl-files's URL and an user friendly name to later reference the generated Web Service's stub. The wsdl-file has to be processed by a stub-generator such as wsdl2java from the Apache Axis project. The programming-language used for the implementation has to support late binding to allow dynamical adding of new stubs (classes or libraries) during runtime.

F2: **Create a new object** (instance of a class), such as a Web Service stub

- F2.1: Specify a class an instance should be created from (usually pick one from a list)
- F2.2: List all constructors of this class and their required parameters to invoke a constructor. This requires a mechanism to enable the extraction of interface-definitions. Java-reflection for example allows easy extraction (such as method names and parameter types) from compiled classes. However, names of the parameters and further meta-information (usage description) have to be supplied separately. Unfortunately this meta-data cannot be extracted from UDDI because its entries are not fully machine-readable
- F2.3: Select a constructor to invoke
    - F2.3.1: Supply parameters to invoke constructor – parameters are:
        - F2.3.1.1: references to existing objects (i.e. referenced by their name)
        - F2.3.1.2: newly created objects (see F2)
        - F2.3.1.3: native objects (created implicitly, i.e. a number or string-literal)
- F2.4: Invoke the constructor
- F2.5: If the constructor's invocation succeeds a new object has been created. This can either be used directly as described in F3.2 or a name can be assigned to the object to allow a later usage as a reference (see F3.1)

F3: **Use an object created with the mechanism described in** F2

- F3.1: To use an object it's assigned name (see F1, F2.5) has to be specified (usually by picking one out of a list)
- F3.2: Using meta-data (i.e. generated using reflection-mechanisms, see F2.2) a list is created containing all methods with their required parameters
- F3.3: Select a method to invoke
    - F3.3.1: Supply parameters: these can be specified as described in F2.3.1
- F3.4: Invoke the method
- F3.5: If the call succeeds and the method has a return value it can be used directly as described in F3.2 or a name can be assigned to the object to allow a later usage as a reference (see F3.1)

The functionality described above is nothing new and can be found in many existing implementations that provide online Web Service tutorials like XMethods.com [8]. However these solutions are primarily for educational or explorative Web Service usage and are not intended for frequent usage. Many implementations are designed beginner-friendly which results in long wizard-driven mouse-click-streams. Saving a query or its results is usually not possible and many implementations support usage of a single Web Service only, meaning that join operations or sub-queries using multiple Web Services are not possible, i.e. if one Web Service returns an address and another Web Service uses the address's zip code to lookup specific data for that region.

We want to introduce a keyboard-controlled approach using a browser-embedded query-console. Keyboard-controlled usage has advantages for frequent users. There is no mouse-keyboard switching between data-entry (usually with a keyboard) and option selection (usually with a mouse), navigation is menu-driven via cursor-keys or shortcuts. This speeds up usage. The downsides are all disadvantages from shells (DOS-prompt) and early terminal applications: they are not intuitive and skills are required. Non-experts have to read manuals perpetually because commands and their parameters are easily forgettable.

The '*what can I type here*' problem in keyboard-controlled environments has already been solved for environments with a defined syntax. Modern IDEs feature source-code-completion by analyzing the current source-code, matching it with syntactical-rules, looking up meta-data for the current context and finally presenting a list of valid method-calls, object-references and other meta-data such as help-texts, etc.

In our scenario the current source-code corresponds to the query, the syntax-rules are well known (see fig. 1 for a simplified version) and relevant context information can be looked up easily using Java-reflection mechanisms. This allows auto-completion for Web Service based queries making skills optional and preventing shell-typical disadvantages as described above.

We realized a prototype with the functionality as described above. Our solution is a keyboard-controlled and browser embedded console that features auto-completion for class- and object-selection (due to too many alternatives for a select-box) and a menu-driven selection mechanism for method- and constructor selection. The prototype has been realized using Java with Apache AXIS. XML-documents are generated from meta-data (reflected class interface information) for the current context. These documents are transformed to HTML using XSLT-transformations. The HTML-output is being processed by a Java-servlet.

4 Automation

The query-mechanism described above already allows faster usage of services with greater flexibility than common HTML-forms. However, several scenarios like automatic execution of such queries or comparing their results is not covered. In this chapter these automation-scenarios are to be addressed. They are:

- **sequences of operations:** execution of multiple queries
  i.e. get price of item I, get price of item P
- **conditional expressions:** if-then-else statements
  i.e. if price of I < price of P then buy I
- **scheduled operations:** automatic execution at predefined times
  i.e. at 12:00 daily: check stocks
- **event-triggered operations:** automatic execution at predefined events
  i.e. on parcel arrival send mail
- **set-based operations:** execution of for/for-all statements
  i.e. for all stocks in watchlist compare prices

Today, these scenarios are out of scope for non-programmers and they are too inconvenient to specify for many programmers, due to having to write and maintain a special program for them.

A solution for programmers is fairly simple: our approach is to bring the IDE-concept to the web by embedding a JavaScript editor with code-completion into a web-browser. Helper-methods ease stub-generation for Web Service usage, scheduling of method-invocation and event handling. Hereby simple automation-scenarios (short programs) can be specified by programmers without the need of prior software-installation/configuration assuming this IDE is offered by a service provider.

For non-programmers this IDE is still unusable due to the need of having to learn a programming-language. This problem is well-known and as old as programming itself, therefore a variety of concepts exist to abstract from this complexity, such as so called *easy-to-learn* functional languages like BASIC, descriptive languages like SQL (originally designed for managers), and visual (do not mix up with symbolic) languages [9]. Mapping a programming-language to graphical symbols (and reverse) is trivial – just visualize the parser's result (usually a tree). The challenge is finding a representation that is useful, such as one that can be understood easily.

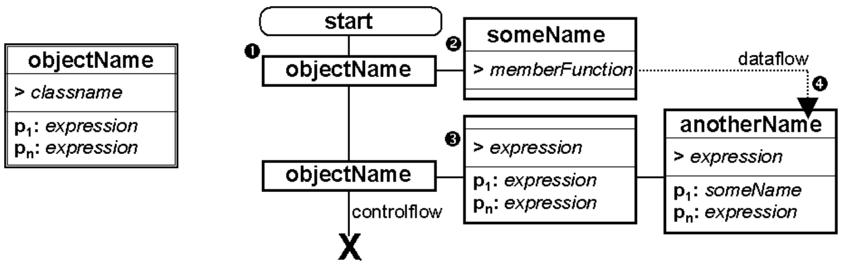


Fig. 2. Example of how to map the simple statements from fig. 1 to graphical symbols

To enable simple, spontaneous usage of Web Services we introduce the representation as depicted in fig. 2 that is tailored to our needs as described in section 2. This example visualizes the following sequence of simple JavaScript statements.

```
objectName  = new classname(p1, p2, p3);
someName    = objectName.memberFunction();
anotherName = objectName.someFunction(p1, p2, p3).
              anotherFunction(someName, p2);
```

Instantiation of classes and Web Service stubs is represented using a double-bordered box: *objectName* is the name of the assigned variable, *classname* specifies the name of the class to be instantiated and $p_1,...,p_n$ is the *parameterlist* (see fig. 1) required to call the constructor of the class. Objects are referenced by their name, see ❶ and can be used to call member-functions, see ❷: *memberFunction* defines the name of the method to call and *someName* is the optional name of the variable its return-type is assigned to. If the member-function or alternatively any expression requires parameters they are specified using a list, see ❸. If a parameter is a reference to an object, dataflow is indicated by a dotted line, see ❹. A solid vertical line indicates the control-flow. Graphical symbols for execution scheduling, event-subscription, set-operations and conditional expressions are described in appendix A.

Symbols like these are easier to learn than a programming-language; however the user still does not know how to combine them. Instead of a drag&drop-GUI we recommend the usage of context-sensitive menus known from code-completion. Clicking somewhere opens a menu with *what can I do here* alternatives, such as change a parameter or call a method. This ensures that modifications to the visual-model comply with a syntactically and semantically correct JavaScript program at all times.

Experiments with sample Web Services revealed that even simple services require multiple instantiations and method-calls even for simple use-cases. The context-sensitive menus as described above do not explain the service and API-documentation – if available – is targeted for programmers only. Therefore, we use an automatically generated documentation using graphical symbols that are a simplification of an UML class-diagram to visualize coherence between classes and their methods.

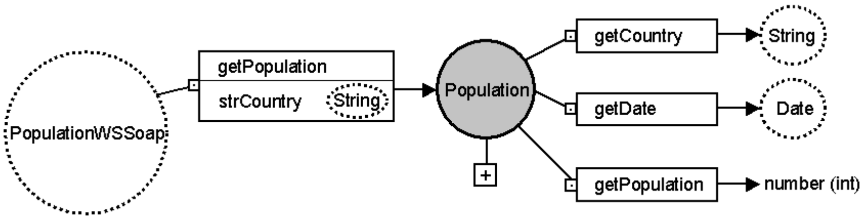


Fig. 3. Example of how to document a Web Service using graphical symbols

Figure 3 depicts how an existing Web Service for querying the population of a country is documented using our graphical symbols. Classes (*Population*) are represented using a circle, its methods by boxes (such as *getPopulation*). Required parameters are displayed as a list of parameter-name and type. The return-value of a method is depicted by an arrow. To avoid complex class-diagrams, some methods are omitted. These can be included by clicking the plus-icon. Another way to keep the diagram small is using references. These are depicted by dotted-circles, such as *Date*. Clicking on *Date* shows the documentation of the date-class.

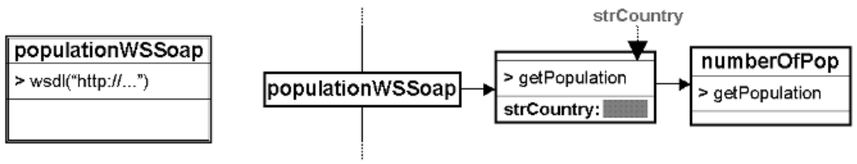
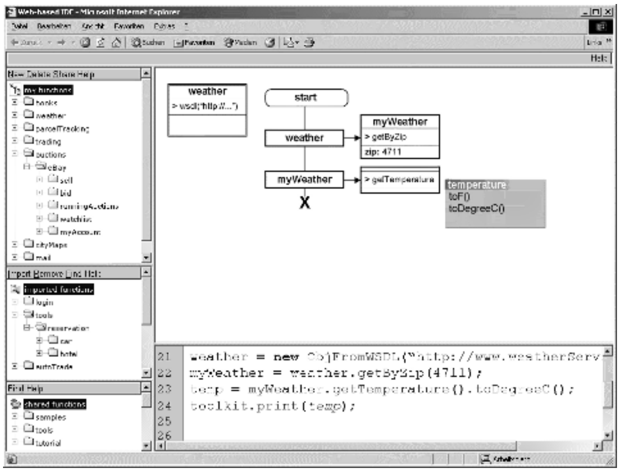


Fig. 4. Generated method-call for getPopulation()

This documentation allows the user to interactively explore a Web Service. It is also used to automatically generate method-calls for the user. Clicking on the getPopulation method (the one without parameters) generates a method call as depicted in figure 4. Required parameters (the name of the country) are marked red and can be specified by the user either by entering a constant or the name of an object to reference it. The documentation mechanism described above can also be used for reverse-search if you need to know where a String to specify the name of the country comes from all methods returning this type can be listed. To keep this list short only methods of the current package are shown.



```
weather = new ObjFromWSDL("http://www.weatherServ
myWeather = weather.getByZip(4711);
temp = myWeather.getTemperature().toDegreeC();
toolkit.print(temp);
```

Fig. 5. Screenshot of the web-based IDE (integrated development environment) prototype

We currently realize a prototype (see fig. 5) with the following functionality. On the right side both, the graphical model and the JavaScript code can be edited, changes are synchronized between them automatically. The JavaScript editor uses DHTML, the graphical editor either SVG or generated images. On the left side folder-trees are displayed for three categories. The first (upper-left) shows the methods created by the user, the last shows shared methods from other users and the middle shows imported methods picked from the shared-list. Shared-methods enable reuse and can be used as a tutorial/reference by beginners.

Security Issue

Warning: If you are interested in providing a service as shown above, take precautions that users can only use a safe subset of classes (such as not *java.io.File*), that programs always terminate (limit iterations of for-loops) and that programs can not be used as spam-generators (limit amount of mails sent).

5 Related Work

Bringing an IDE to the web does not include a feature which is big, important, or completely new. However, it is an interesting new combination of a selection of known techniques customized for a new target group of experienced web users. In this section we will describe relevant concepts used by existing applications.

HTML-Form based Interfaces: There are a number of generic clients available on the Internet which allow interacting with existing Web Services. However, those interfaces are mainly for programmers that only explore a service. Form based interfaces are not built for a frequent usage and do therefore not offer automation or even saving functionalities.

Dynamic and adaptive composition of services: Some systems support a graphical composition of Web Services. Most often, those systems have their origin in the processes world and are mainly made for modeling processes. Code fragments can be generated out of the process-models. One example is eFlow [10] which supports the dynamic generation of Web Services sequences. Graphical front-ends for BPEL4WS can also support this job.

Graphical Entering of Workflow: There is a variety of tools which support graphical modeling of process representations. These can be activity diagrams as used by the Unified Modeling Language, UML, or Petri Nets which have been used for many years now. Such a graphical support helps especially novice and non experts to do their work and enter the processes in a syntactically correct way.

IDE – Code Completion: Almost all modern integrated development environments such as eclipse offer a feature commonly known as code completion. On entering the name of an object to call one of its methods, a list box containing all methods which match the so far entered name is generated and displayed.

Code Sharing: The generated code and history of executed steps during the use of our concept can often speed up future uses. This benefit can be made available to other developers, too. They can use the generated information and functions like libraries of programming languages which are also often shared by programmers. The same idea has successfully been applied to many open source projects and huge parts of the libraries of many programming languages have been developed this way.

MDA: An important concept of model driven architectures, MDA, is the use of a model as central element. The required code is generated from the information of the graphical model and modified to execute additional jobs. If the architecture of the application has to be changes, only the model is modified and the code is regenerated. Therefore, the documentation is always up to date, because the model illustrates the architecture and is source for the generation of the code at the same time.

6 Conclusions

In this paper we argued that the trend of many web sites to open their API by providing Web Service interfaces could be used not only to satisfy the need of application programmers but also enable end-users that use web sites prevalently to better support their usage scenarios by enabling efficient query and automation mechanisms.

This group of ‘experienced web users’, as we call them, wants to perform actions on web sites which are not well supported by the site’s graphical front ends. Unfortunately, the Web Service interfaces provided by many web sites today do also not meet the requirements of this group of users. Although those interfaces provide the needed functionality for extended usage, two restrictions impede their application: First, even experienced web users do not want to use a fully fledged programming environment. Since it has been designed for general programming tasks it is much too complex for the simple query and automation tasks this kind of users want to perform. Something more simple is needed that enables a steep learning curve with short time to productive usage. Second, a runtime environment is required at the users workstation. Most of the experienced web users do neither have the skills nor the resources nor do they want to spend the efforts of installing and operating such an environment. For this reasons we have proposed to provide such an environment as a web-based service and to reduce the functionality of this service to the core programming elements.

In section 2 we have described the requirements such a service has to fulfill: It has to abstract from Web Service complexity (SOAP, HTTP-connection, bindings, endpoints) and programming-complexity (classes, functions, control-structures). By this we ensure that it is easy to use. Further we introduced a reduced subset of programming elements which we think is sufficient for supporting the tasks of a experienced web user. Hence, the user does not need to cope with the complexity of programming with Web Services in a conventional IDE.

In the following sections 3 and 4 we presented two typical usage scenarios and how they are supported by the web-based environment we propose. To meet the last requirement introduced in section 2 (avoid local installations) our solution for both scenarios is web-based and uses a browser as a front end.

The first scenario is to perform queries which may span several Web Services. Well-known query languages, e.g. SQL, are neither able to cope with the service-oriented interfaces nor with heterogeneous data sources nor with semi-structured data. Nevertheless, in our opinion such queries can be expressed in a function-oriented programming style with nested service calls and this way of expressing queries can easily be mapped to a web-based query console. Since this is only a subset of the second scenario we described the web-based programming environment in section 4.

The second scenario extends the query console by additional programming elements, e.g. conditional execution. In section 4 we have shown how a steep learning curve is made possible through a mixture of graphical and textual programming. In addition, the meta data available for Web Services enables us to support the user by context-sensitive suggestions about what to do next. In an example we demonstrated that even non-programmers can use this interface and become productive fast.

Altogether, the main messages of our paper are:

- There exists a large group of users which is not supported well so far, i.e. there is a reason to provide such a service.
- It is possible to provide a service which does meet the requirements and restrictions of this group of users, i.e. there is no reason not to implement such a service.
- The Web Service technology available today is sufficient to implement this kind of service, i.e. there is no reason not to start now.

Nevertheless, one question remains open: Why is no such service available?

References

1. Konopnicki, D., Shmueli, O.: W3QS: A Query System for the World Wide Web. 21st Conference on Very Large Databases, Zurich, Switzerland, 54—65, 1995
2. Levy, A. Y., Rajaraman, A., Ordille, J. J.: Querying Heterogeneous Information Sources Using Source Descriptions. In proceedings of the 22nd VLDB Conference, Bombay, India, 1996
3. Ashish, N., Knoblock, C. A.: Semi-Automatic Wrapper Generation for Internet Information Sources. Conference on Cooperative Information Systems (1997) 160-169
4. eBay news
http://presse.ebay.de
5. Axis: Apache Web Services Project
http://ws.apache.org/axis/
6. JavaScript ECMA Standard: ECMA-262
http://www.ecma-international.org
7. Extended Backus-Naur Form (EBNF), ISO/IEC 14977 (1996)
8. XMethods Web Services collection
http://xmethods.com
9. Burnett, M. M.: Visual Language Research Bibliography
http://web.engr.oregonstate.edu/~burnett/vpl.html
10. Casati, F. , Shan, M.-C: Dynamic and Adaptive composition of E-services. Information Systems (2001) 26:143–163

Appendix A Graphical Symbols

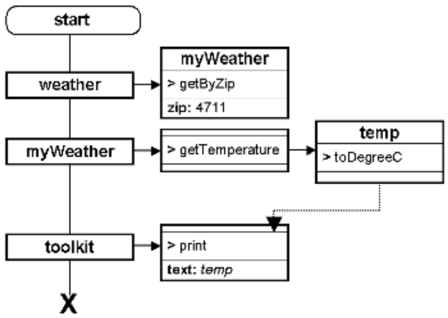


Fig. A.1 Sequence

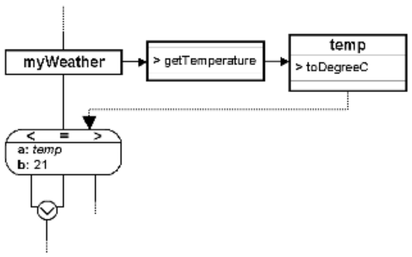


Fig. A.2 Simple conditional expression

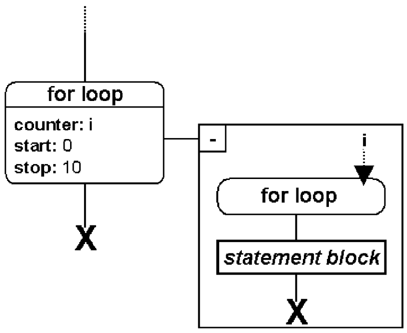


Fig. A.3 For-loop

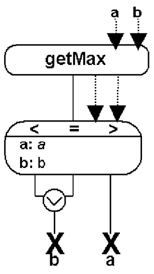


Fig. A.4 Method declaration with parameters and return-type

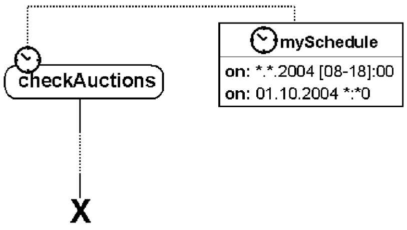


Fig. A.5 Execution scheduling

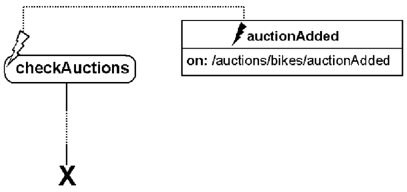


Fig. A.6 Event Subscription using a topic-path expression from web-services notification

SOAP Processing: A Non-extractive Approach

Jimmy Zhang

2051 Grant Rd, Los Altos, Ca 94024, USA
jzhang@ximpleware.com

Abstract. As the first step of most XML processing algorithms, one usually extracts token content out of the source document into many discrete string objects. We propose a "non-extractive" tokenization approach that maintains the source document intact in memory. Using a binary encoding specification called Virtual Token Descriptor (VTD), the processing model represents tokens exclusively using starting offset and length. To create a hierarchical view of the data encapsulated in the SOAP message, the parser further indexes elements of same depths using directory-like structures we call location cache. Through a demonstration of navigating the document hierarchy using VTD and location caches, we show that it is indeed possible to create a cursor-based API that retains most of DOM's random-access capabilities at a fraction of its memory usage. Furthermore, by analyzing key design constraints of custom hardware, we reason that the memory conserving characteristics of the processing model simultaneously make possible "SOAP on a chip" and "binary-enhanced SOAP." The benchmark results show that the reference implementation of our processing model significantly outperforms Xerces DOM in terms of both memory and processing performance.

1 Introduction

SOAP (Simple Object Access Protocol) is the W3C standard that defines a way to encode data using a subset of XML. Web services, the next generation middleware technology, choose SOAP as the wire format in order to overcome interoperability issues associated with prior-generation middleware technologies, e.g. CORBA and DCOM. As more enterprises deploy Web Services as a primary way to integrate applications, SOAP traffic is increasingly rapidly in the network. This means that application servers are processing more SOAP messages; network firewalls and intermediaries are routing and validating more SOAP messages. Although SOAP is being handled in so many different ways, one thing remains the same: A SOAP message needs to be parsed before anything else can be done with it. Because SOAP is basically a subset of XML, in the rest of this paper, we focus on the processing of XML. Inheriting heavily from traditional text processing techniques, existing XML processors extracts tokens out of the source document into many discrete string objects. Subsequently, one can build data structure or class hierarchies on top of those tokens.

Nevertheless, we would like to observe that, to achieve the purpose of tokenization, one has another option, i.e. to *only* record the starting offset and token length, while leaving the token content "as is" in the source document. In other words, we can treat the source document as a large "token" bucket, while creating a map detailing the positions of tokens in the bucket.

To help illustrate how this "non-extractive" style of tokenization works, we compare it with traditional "extractive" tokens in some common usage scenarios:

- **String comparison.** Under the traditional text-processing framework, one uses some flavors of C's "*strcmp*" function (in <string.h>) to compare an "extractive" token against a known string. In our approach, one simply uses C's "*strncmp*" function in <string.h>.
- **String to numerical data conversion.** Other frequently used macros, such as "*atoi*" and "*atof*," can be revised to work with non-extractive tokens. One of the possible changes is the signatures of the functions. For example, "*atoi*" takes a character string as the input. To make a non-extractive equivalent, one can create a "*new-atoi*" that accepts three variables: the source document (of the type char*), offset (of the type int), and length (of the type int). The difference in implementation is mostly to deal with the new string/token representation (e.g. end of string is no longer marked by \0).
- **Trim.** Removing the leading and trailing white spaces of a "non-extractive" token only requires changes to the values of offset and length. This is usually simpler than extractive style of tokenization, which often involves the creation of new objects.

Overall we feel that, apart from some implementation changes, the difference between traditional tokenization and the proposed "non-extractive" tokenization is largely a change in perspective. They are just two different ways to describe the same thing, i.e. a token.

Going one step further, one can implement various "non-extractive" functions to directly deal with native character encoding. For XML processing, application developers are more used to UCS-2 strings within their code. Those functions need to be "smart" enough to understand different character encodings and, at the same time, export UCS-2 compatible signatures to calling applications.

Outline. In the ensuing sections of this paper, we build upon the concept of "non-extractive tokenization" in the context of processing the SOAP subset of XML. In Section 2, we introduce Virtual Token Descriptor (VTD) and Location Cache (LC), both of which are designed to enable a cursor-based XML processing model that is described in Section 3. We then present some preliminary benchmark numbers indicative of the processing and memory usage characteristics of the processing model in Section 4. Following the analysis on memory-conserving aspects of the design, we reason, in Section 5, that those design considerations not only have implications on storage of XML, but also help remove a key obstacle to porting XML processing on dedicated hardware. Section 6 concludes this paper by acknowledging some limitations of the processing model.

2 A Processing Model Based on Virtual Token Descriptor

Similar in many ways to how DOM processes XML, our processing model first generates in-memory data representation of XML, then exports it to calling applications through a cursor-based navigation API. To help illustrate different components of the data representation, we further divide the first step into (1) "non-extractive" tokenization using Virtual Token Descriptor (VTD) (2) building hierarchical element directories using Location Cache (LC).

In the rest of this section, we introduce the concept of each component, and then describe some properties of the processing model.

2.1 A Quick Overview Virtual Token Descriptor and Location Cache

While a tokenization approach using offset/length is adequate for processing unstructured text files, one needs additional information to describe a token in an XML file. This has led to XimpleWare's design of VTD (Virtual Token Descriptor). Unlike DOM or SAX, VTD is not an API specification; it is instead a binary format specifying how to encode various parameters of a token. A VTD record is a primitive data type that encodes the starting offset, length, type and nesting depth of a token in XML. For certain token types, we further split the length field into prefix length and qualified name length since both share the same starting offset. VTD records are usually stored in chunk-based buffer in the same order as they appear in the document. In addition, the processing model built on top of VTD mandates that one maintain the original document intact in memory.

Specific to our current implementation, a VTD record is a 64-bit integer in network byte order (big endian) with the following bit-level layout:

- **Starting offset**-30 bits (b29 ~ b0)
- **Length**-20 bits (b51 ~ b32)
- **For some token types**
  - ❑ **Prefix length**: 9 bits (b51~ b43)
  - ❑ **Qname length**: 11 bits (b42 ~ b 32)
- **Nesting Depth**-8 bits (b59~b52) -- Maximum value is 2^8-2 = 254
- **Token type**-4 bits (b63~b60)
- **Reserved bit**-2 bits (b31: b30) are reserved for a tri-state variable marking namespaces.
- **Unit**--Because the processing model doesn't decode XML, the unit for offset and length are in raw character of the transformation format. For UTF-8 and ISO-8859, length and offset are in bytes. They are in 16-bit words for UTF-16. Fig. 1 depicts the bit-level layout of a VTD record.

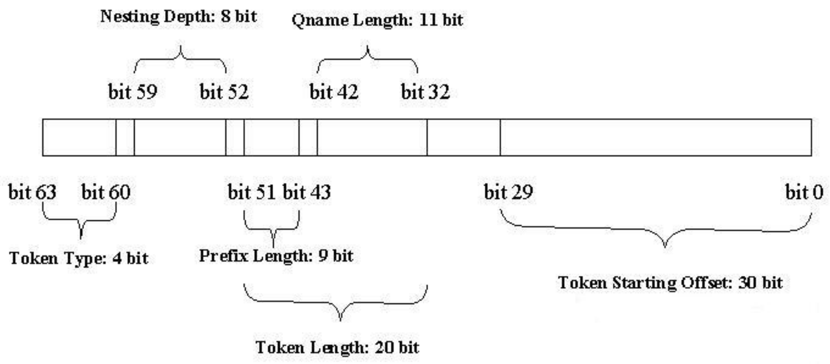


Fig. 1. Bit-level layout of a VTD record.

To create a hierarchical view of XML, we collect addresses of elements (i.e. VTD records for starting tags) into chunk-based structures we call Location Caches (LCs). Location caches are allocated on a per-level basis; i.e., the same LC indexes all elements of the same depth. Similar to a VTD record, a LC entry is also a big-endian 64-bit integer. It upper 32 bits contain the address of the VTD record. The lower 32 bits point to the LC entry corresponds to the first child element. If there is no child element, the lower 32 bits are marked with -1 (0xffff ffff).

At the top level, the parsed representation consists of the components shown in Fig. 2 after the parser finishes processing XML. Notice that, in addition to aforementioned components, we also include RI, the address of the root element, which doesn't change once those components have been generated. The same applies to M, the maximum depth of the document, which determines the size context object (discussed in Section 3) for subsequent navigation purposes.

2.2 Location Cache Lookup

At the top level, the LCs are essentially hierarchical element directories of XML. The upper 32 bits of an LC entry convey the "directory" aspect of LC as they explicitly associate an LC entry with an element. The lower 32 bits, on the other hand, convey the "hierarchy" aspect as they bind the LC entry with the one corresponding to its first child element. To look up the VTD index of a child element in LCs, one usually goes through the following steps: (1) Record the LC entry of the first child. (2) Calculate and record the LC entry of the last child. (3) If the lookup is to find the first child, move the "current" LC entry over to point to the first child. Fig. 3 illustrates various values and locations recorded after the lookup for the first child element. Notice that the new location of current entry is colored in red in Fig. 3.

For the second step above, the LC entry of the last child is calculated as follows: (1) Find the "child-ful" LC entry immediately after the current one. (2) Retrieve the lower 32 bits of that entry (k). (3) The LC entry of the last child is the one immediately before k. Notice that, in Fig. 3, the entry immediately after the current one is skipped because it is childless (marked by X).

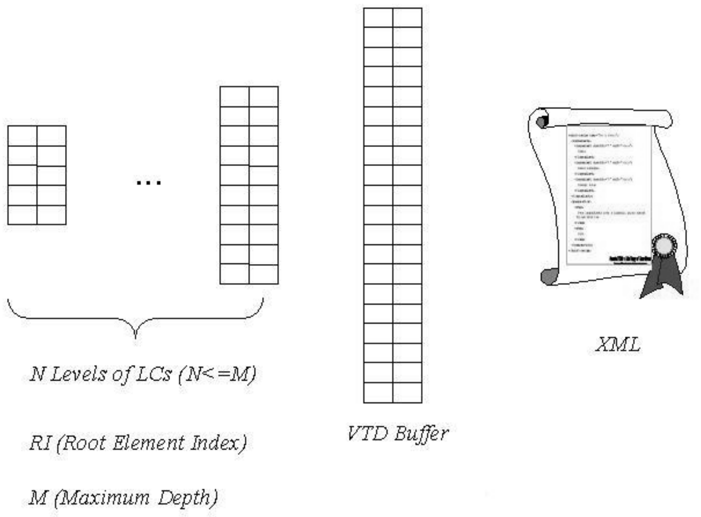


Fig. 2. Parsed Representation of XML.

After those lookup steps, one can be sure that the segment of the LC entries delimited by the first and last child covers every child element of the current LC entry. If the next lookup is to find the next sibling, one simply move the current entry down one unit, the use the upper 32 bits to locate the corresponding VTD record.

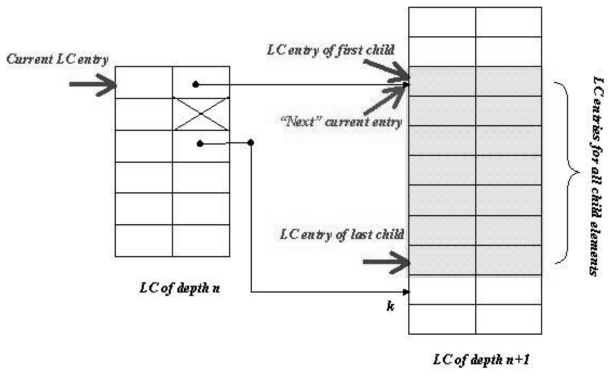


Fig. 3. Resolving child elements using Location Cache.

2.3 Properties of the Processing Model

We highlight some of the noteworthy properties of the processing model.

- ***Keep XML intact in memory.*** As the parsed state of XML, VTD records per se don't carry much meaning; they only become meaningful when used alongside of the source XML document. VTD records describe the structure of XML and provide access to the token contents. If the application needs those tokens, it still has to pick them up from the "bucket."
- ***Use integers, not objects.*** The parsed representation of XML makes extensive use of 64-bit integers at the lowest granularity level. Both LC and VTD use 64-bit integers as basic information storage units. This has both pros and cons. A noticeable con: A VTD record doesn't have member methods. One of the pros is that it is platform independent: Architecture-specific implementations must explicitly conform to binary specifications all the way down to bit-level layout.
- ***Compact encoding.*** Both VTD and LC strive to squeeze maximum amount of information into every record/entry. Without either one of those four parameters (offset, length, type and depth), a VTD record becomes far less effective in describing tokens of XML for subsequent navigation purposes. In some cases, a single VTD record describes up to two tokens (the prefix length and qualified name length). For a LC entry, the lower 32 bits point to the first child element. The last child is inferred from another entry (usually the one right after) of the same LC.
- ***Constant record/entry length.*** In a linked list, individual members are associated with each other by the explicit use of pointers. Those members must have pointer fields defined as the instance variables. Because VTD records and LC entries are equal in length (64-bit), when stored in chunk-based buffers, related records/entries are associated with each other by *natural adjacency*. For example, an element token is usually near its attributes. Also the next sibling, if there is one, is always immediately after the current LC entry.
- ***Addressable using integers.*** As a basic requirement, one should be able to address a token/entry after parsing, e.g. to build a hierarchy. VTD records, when stored in chunk-based buffers, can be addressed using indexes (integers). This is different from extractive tokens, which can only be addressed by pointers. As a result, many functions/methods in the navigation API returns an integer that addresses a VTD record (-1 if no such record).
- ***Element-based Hierarchy.*** Unlike DOM, which treats many node types as parts of the hierarchy, the processing model builds hierarchy for elements only. VTD's natural adjacency allows one to locate other "leaf" tokens, e.g. attributes or text nodes, via direct searches in the vicinity of element tokens. For mixed-content XML, LC entries provide additional hints for possible locations of "free-form" text nodes.

In addition, there are some practical considerations concerning the layout of a VTD record. First, both prefixes and qualified names for starting tags and attribute names usually are not very long. The maximum allowed lengths, 2048 (11 bits) for qualified name and 512 (9 bits) for prefix, should be sufficiently for any practical uses.

Additionally, we try to be conservative in our choice of encoding the nesting depth in an 8-bit field (up to 256 levels). Most of files we have come across, especially large ones, are usually more flat than deep. Our VTD parser throws an exception if there is an overflow condition for the prefix length, the qualified name length or the depth of a token. For other token types, such as character data, CDATA and comment, the VTD parser can potentially handle length overflow by using multiple VTD records to represent a single XML token.

2.4 Memory Usage Estimation

We would like to make two assumptions before estimating the total memory usage. First, we assume that every starting tag has a matching ending tag. Secondly, VTD records for ending tags are discarded because they carry little structural significance. If there are a total of $n1$ tokens (including ending tags) and $n2$ elements (starting tags) in a document of size (in bytes) s, the total size of VTD records in bytes (without ending tags) is $(n1 - n2) \times 8$ and the total size of LCs (totally indexed, i.e. one LC entry per element) is $n2 \times 8$. The sum of all those components is: $(s + 8\times (n1-n2) + 8\times n2) = s+8\times n1$.

As we can see from the calculation, the memory usage is a strong function of the token "intensity" of XML, and basically not affected by the structural complexity of the document. For document-oriented XML files that have lots of text, e.g. a novel, our experience has been that the memory usage is around 1.3 times the size of the document (assuming UTF-8 encoding). For data-oriented XML documents, the multiply factor is typically around 1.5~2.

3 Navigating XML

After the parser finishes processing XML, the processing model provides two views of the underlying XML data. The first is a flat view of all VTD records. Corresponding to all tokens in XML in document order, it can be thought of a view of cached SAX events. The second is a hierarchical view enabled by a cursor-based navigation API allowing for DOM-like random access within the document. And the cursor always points to the VTD record of the current element. In this section, we first describe the important concepts and internal constructs of the navigation API, then demonstrates how everything works together to achieve the purpose of navigation.

3.1 Context Object

To navigate the element hierarchy represented by VTD and location caches, the processing model first creates an integer array (whose size equals the maximum depth M) we call Context Object (CO). Its primary purpose is to track, at any given moment of navigation, the position of the current cursor in the element hierarchy. We use the first entry (CO[0]) in the context object to indicate the current depth of the cursor. The rest of the array is laid out as follows: Assuming the current depth of the cursor is

D (D <= M), we place the VTD indexes of the cursor and all its ancestors (except root index) into the context object according to their depth values (CO[1] ~ CO[D-1]), as shown in Fig. 4. Any unused entries are filled with -1.

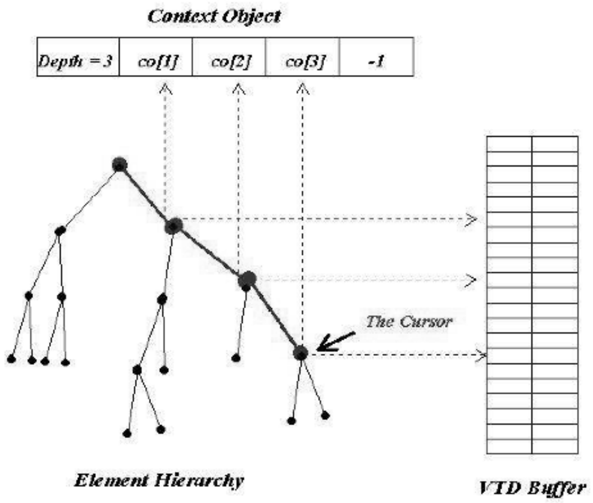


Fig. 4. Mapping element hierarchy into Context Object.

Additionally, the context object maintains the necessary state when one wants to look up the namespace URL for a given prefix. This is done by a bottom-up search starting from the attribute tokens under the current element. If no match is found, move up one level and look for attribute tokens of the parent. Repeat this process until one finds the matching name space token and the URL value.

3.2 Element Indexing Strategy

While the context object maintains the necessary internal state for navigating the element hierarchy, it does not fully specify how to obtain those VTD indexes from the VTD buffer and LCs. Indeed, the element indexing strategy of the processing model strongly influences the internal behaviors of the navigation. We summarize those strategies and corresponding navigation behaviors below:

- ***VTD scan without LCs.*** With this option, the processing model doesn't allocate or use LCs at all. Because a VTD record encapsulates both its depth and type, one simply obtains the VTD index of its sibling and children by directly scanning the VTD buffer. Scan backwards if one wants to find the previous sibling; scan forward for the next sibling, the first child and the last child. In fact, the processing model emulates the behavior of Xerces' nodeIterator by scanning VTD buffers for element tokens and simultaneously updating the content of Context Object. The performance is bound by the maximum bandwidth of the memory. For SOAP header processing purposes [6], this option provides an important capability that a

SAX parser lacks by default, i.e. being able to go through the same content multiple times without reparsing.

- ***Full element indexing.*** The processing model can choose to index all elements in XML, i.e. one LC per depth level up to the maximum document depth. In this case, VTD indexes are entirely looked up from LCs.
- ***Proximity element indexing.*** For this, we can heuristically pick a depth level so that navigating below that level is by LC lookup; above that level, scan the VTD buffer. We find this strategy suitable in many situations for several reasons. First, full element indexing incurs the cost of allocating LCs and indexing element. Secondly, scanning the VTD buffer, effectively a large number of sequential memory reads, is precisely what the current generation of memory technologies (e.g. DDR SDRAM) are best at. For example, Micron's Samurai DDR chipset (64-bit data bus) [8] running at 133 MHz delivers maximum throughput of over 2GB/s. Furthermore, since one of our design goals is to move the processing model on chip (more discussion later), in order for custom hardware to process XML in a single pass, we find it much easier to hardcode levels of LCs in silicon. Full element indexing in hardware is difficult because the hardware is less flexible and lacks a prior knowledge of the depth of XML. Our initial software implementation employs this proximity indexing strategy with a pre-chosen value of 4.

3.3 An Example

In Fig. 5 we show the content of the context object during a typical navigation run. VTD records and their corresponding tokens are highlighted and labeled with their VTD indexes. We use green color to indicate the element tokens since the cursor only touches those during navigation. Initially, the cursor points to the root element. As it moves deeper into the hierarchy, the depth value and the corresponding entry (underscored ones in Figure 4) are getting updated to reflect the position change of the cursor. To move up the hierarchy, we only need to update the depth value and the corresponding entry in the context object.

3.4 Important Methods

In our reference implementation, we organize various parsing and navigation functions into two classes:
VTDGen and VTDNav. VTDGen takes as the input an in-memory byte array containing the XML document, and produces VTD records and location caches. VTDNav contains the member methods that navigate the hierarchy, compare tokens to strings, and convert tokens to strings or numerical data. In Table 1, we list some of the important member methods of VTDNav.

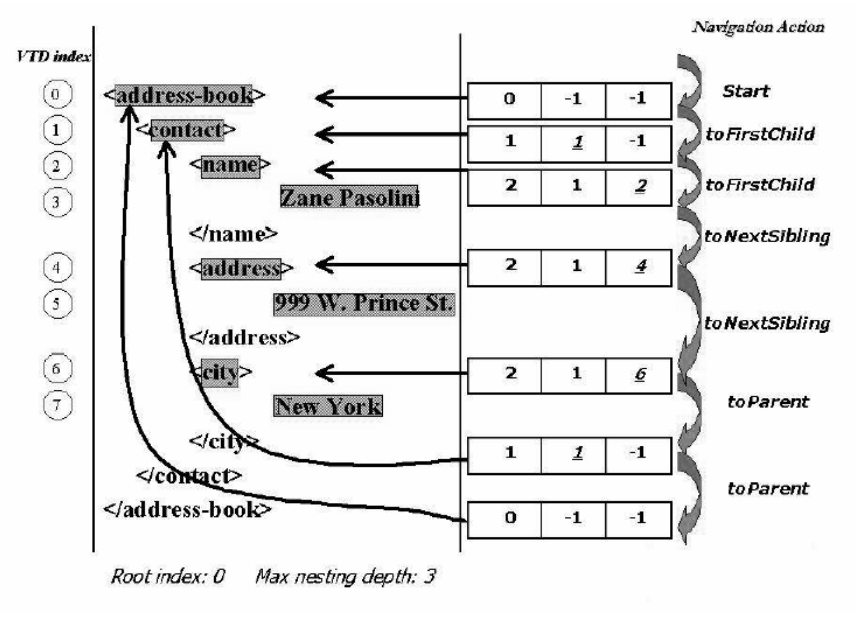


Fig. 5. Navigating XML using VTD.

Table 1. Selected Member Methods of VTDNav

Method Name/Signature	Description
Boolean toElement(int direction)	This is the primary function for element hierarchy navigation.
Boolean toElement(int direction, String ElementName)	Same as last one, plus one needs to specify element name
Int getText()	This function returns the VTD index for the text node of the current element
Int getAttributeValue(String AttributeName)	This function returns the VTD index of an attribute value for a given attribute name
Int parseInt(int index)	This function takes a VTD index and returns the integer value represented by the token content

4 Latest Benchmark Numbers

We benchmarked our reference implementation written in Java using 1.42 version of JVM on an Athlon 1800+ machine running Windows 98. The parser is non-validating and supports the SOAP subset of XML. It parses the DTD, but doesn't attempt to resolve entity declarations. Most of our implementation effort has gone into improving the performance of VTD generation, which, according to our observation, dominates the overall processing. The primary focus of our benchmark is to compare various aspects of our processing model with Xerces DOM 2.3 (via JAXP). We also selectively include some performance figures of Xerces SAX as another reference point in the comparison. Namespace awareness is turned off for all tests.

4.1 VTD Generation

Table 2 shows the performance numbers of processing a set of XML documents whose sizes range from small (30k) to very big (>20MB). For some reason, Xerces doesn't perform well for small documents. When files get bigger (>200kB), Xerces' performance starts to get better. In contrast, VTD's processing performance is more linear.

Table 2. Parsing Performance Comparison

File Name	size (kb)	DOM(ms)	VTD(ms)	SAX(ms)
ab.xml	31.7	12.1	1.31	2.2
po100k.xml	101.6	25.9	3.41	5.06
soap.xml	1255.04	280.1	66.3	67.03
po1m.xml	1014.8	181.2	41.2	42.6
ot.xml	2523.4	246.1	61.1	53.8
bioinfo.xml	18484.4	4290	689.4	977.8
SUAS.xml	13762.6	3402.1	537.6	903.5
bench.xml	21050.1	3808	641.15	850.9

Table 3. Memory Usage Comparison

File Name	File size (kb)	DOM (kb)	VTD (kb)	SAX (kb)
po1m.xm	1014.7	5636.3	1654.6	690
soap.xml	1255	8792.5	2649.7	662.5
ot.xml	2523.4	9741.4	3363.3	189.4
SUAS.xml	13762.6	88786.8	25689.1	412.1
bioinfo.xm	18484.4	134742.1	30350.1	618.2
bench.xml	21050.1	99250.5	32363.3	597.6

Table 3 shows the memory usage comparison between DOM, VTD and SAX. Notice that SAX consumes the least amount of memory-- a result of not building in-memory representation of XML. For this test, we only use files that are larger than 1MB.

4.2 VTD Navigation Performance

To compare navigation performance, we test the total CPU time used to navigate through entire document using depth first traversal, shown in Table 4. It is equivalent to XPATH expression "//*." Although we don't recommend anyone using this XPath expression in their applications (too processing intensive), we think it provides a good quantitative view of the navigation performance of our processing model. Our observation is that for an XML file containing a large number of attributes, VTD navigation takes twice the time of DOM. For complex structured documents, VTD sometimes is a little faster.

Table 4. Navigation Performance Comparison for //*

File Name	DOM (ms)	VTD (ms)
ab.xml	0.269	0.582
po100k.xml	1.38	2.3
po1m.xml	17	20.8
cd2.xml	2.2	2.8
soap.xml	29.1	32.5
ot.xml	20.9	19.2
SUAS.xml	91.2	186.7
bioinfo.xml	236.7	314.7
bench.xml	280.1	267.4

5 A Closer Look

In this section, we analyze some important aspects of the processing model.

5.1 Memory Usage

The fact that the processing model makes extensive use of 64-bit integers has strong implications on its memory usage, which we summarize as follows:

- ***Avoid per-object memory overhead.*** Per-object allocation typically incurs a small amount of memory overhead in many modern object-oriented VM-based languages. For JDK 1.42, our measurement shows an 8-byte overhead associated with every object allocation. For an array, that overhead goes up to 16 bytes. A VTD record or a LC entry is immune to Java's per-object overhead because it is an integer, not an object.
- ***Use array whenever possible.*** We feel that the biggest memory-saving factor is that both VTD records and LC entries are constant in length and can be stored in array-like memory chunks. For example, by allocating a large array for 4096 VTD records, one incurs the per-array overhead of 16 bytes only once across 4096 records, and the per-record overhead is dramatically reduced to almost nothing. Furthermore, by taking advantages of spatial locality of associated records, one avoids the cost of explicitly stitching together objects using pointers.

Our benchmark results indicate that the total size of the internal representation is usually 1/3 ~1/5 of a DOM tree. For document-oriented XML, the multiplying factor is usually around 1.3, i.e. the internal representation is 30% larger than the document (assuming single-byte character encoding). For data oriented XML, the multiplying factor is between 1.5~2. Comparing with DOM, the memory reduction is most significant when XML is "taggy" and DOM's memory consumption is near the high end of its 5x~10x estimate.

5.2 Inherent Persistence

For applications that demand frequent read-only access to XML documents, the processing model provides the option of saving the internal representations on disk after the XML documents are processed for the first time. Afterwards, those applications only needs to load "pre-parsed" form of XML back in memory. In other words, the in-memory representation of the processing model is inherent persistent. This again is due to the fact that both VTD and LC use 64-bit integers, not objects, to represent tokens and hierarchy. When the processing performance is critical, one can potentially create XML and VTD at the same time, and then package them into a single file before sending it across the network. The downside of this approach is that it will add to the network transmission cost, which may not be suitable for bandwidth constrained environment. The upside is that XML-aware network appliances sitting on the data path can directly use the binary portion of the data to process XML (without parsing again). In addition, the "pre-parsed" form contains the XML text, and therefore loses no inherent benefits of XML, e.g. extensibility and human readability.

5.3 XML on a Chip

Recently, "XML on a chip" has gathered much attention as many companies are racing to deliver ASIC-based XML processing solutions. A big reason is that the performance of XML parsing is often cited as a major bottleneck for many performance-sensitive applications. Matthias et al. [5] identify XML parsing performance to be a key obstacle to a successful XML deployment, and recommend hardware-assisted parsing engine to be a potential research topic. This is hardly surprising, considering the fact that hardware implementations of processing algorithms free CPU from performing processor-intensive computations and sometimes provide the only ways to guarantee QoS. However, one must take into account design constraints that are specific to XML and custom hardware when thinking of hardware-based XML parsing. In that context, our processing model possesses a few attributes critical to the success of implementing "XML on a chip," which we summarize below:

- ***Inherent persistence.*** For the custom hardware to co-process XML and still interface with the application logic, the system most certainly has to transport "something" from the hardware (e.g. a PCI card) into the main address space in which the programs reside. To make the hardware work seamlessly with the application logic, one of the fundamental assumptions of distributed computing applies: The "something" must be persistent because the PCI bus, though carrying a different name, is physically not different than a set of wires like Ethernet, and therefore is subject to the same set of design constraints. Notice that VTD and LCs satisfy this constraint, as both are inherently persistent. Implementing DOM on a chip, on the other hand, is prone to fail because a DOM tree is not persistent.
- ***Constant record/entry length.*** Optionally, the hardware can produce some serialized form of a DOM tree or SAX events. The problem of this approach is that, to rebuild the DOM tree or regenerate SAX events, the software application at the receiving end will have no choice but to go through the

most expensive part of DOM/SAX parsing, i.e., to allocate and garbage-collect objects, all over again, essentially defeating the purpose of hardware-based parsing. The culprit is the variable length of "extractive" tokens. In contrast, because VTD records and LC entries are constant in length and addressable using integers, the persistence of the internal representation is post-binding. In other words, the software application at the receiving end does not incur the cost of allocating a large number of objects, thus reaping maximum benefit of hardware-based parsing.

- ***Minimum buffering.*** In order to obtain maximum performance, a good design should strive to be as stateless as possible. Our processing model, when implemented on chip, features a flow-through, process-then-discard architecture that is quite similar to that of DES (Data Encryption Standard)[11]—an algorithm well known for its high performance hardware implementation. Similar to a DES chip, a VTD chip scans through the entire document one page at a time to record all the relevant information, and no character is accessed more than once. In addition, to avoid the propagation delays associated with accessing off-chip memory elements, the design should keep as many variables/states on chip in fixed-size register files. In that regard, an offset/length based tokenization plays right into the strength of custom hardware.

To sum up, the constant-length token representation is the most important attribute of the processing model. We can also look at this problem from the opposite direction. If tokens are not constant in length, then they are not addressable using integers, and the processing model can no longer bulk-allocate memory buffers to store tokens. Consequently, one probably has no choice but to allocate lots of objects to represent tokens and hierarchy, making per-object memory/processing overhead all but inevitable. And in so doing, applications won't be able to reap significant performance benefit from custom hardware.

6 Conclusion

In this paper we introduced the concept of Virtual Token Descriptor and location cache, both of which are designed to enable a “non-extractive” XML processing model. We also provided a detailed description of the processing model and showed how to navigate the element hierarchy as represented by the combination of VTD tokens and location cache. Attempting to achieve most of DOM’s functionality without incurring its resource overhead, the processing model makes extensive use of 64-bit integers in order to avoid per-object overhead associated with most object-based hierarchies. The benchmark results suggest that we have met most of our design goals. However, we would like to acknowledge the "work-in-progress" status of the work presented in the paper. There are also some limitations of our processing model worth mentioning. First, because VTD makes use of 64-bit integers and fixed-sized fields to encode offset values, for documents that are very large (>1G) or deep, one might need to move bits around, or even add another 32 bits to a VTD record, to meet the actual processing requirement. Second, the current implementation does not resolve entities outside of those five built-in ones (&amps; &lt; &gt; &apos; &quot;).

In addition, our reference implementation doesn't support either DTD or Schema validation. Last, the maximum supported array size in Java is 2G, which is the maximum size that the processing model can handle. As a workaround, we might need to use chunk-based byte buffers to overcome this limit. Finally, we are planning to release the reference implementation under GPL.

Acknowledgements. I personally thank Charley Wang and Harry Xu for useful suggestions and proofreading this paper.

References

[1] *DOM, SAX and JDOM.* http://www.brics.dk/~amoeller/XML/programming/
[2] *XML Applications and Initiatives.* http://xml.coverpages.org/xmlApplications.htm
[3] *XML protocol requirements*. http://www.w3.org/TR/2002/WD-xmlp-reqs-20020626
[4] Adam Bosworth. *Loosely speaking*. http://www.fawcette.com/xmlmag/2002_04/magazine/departments/endtag/
[5] Nicola Matthias, John Jasmi. *XML Parsing: A Threat to Database Performance*. CIKM 2003
[6] Rich Salz. *Processing SOAP header*. http://webservices.xml.com/pub/a/ws/2002/07/17/salz.html
[7] *Apache Axis User's Guide*. http://ws.apache.org/axis/java/user-guide.html
[8] *Micron's Samurai DDR Chipset.* http://www.sysopt.com/articles/samurai/
[9] Chun Zhang, David Dewitt, Jianjun Chen, Feng Tian. *The Design and Performance Evaluation of Alternative XML Storage Strategies*. SIGMOD Record 31(1): 5-10 (2002)
[10] Kenneth Chiu, Madhusudhan Govindaraju, and Randall Bramley. *Investigating the limits of SOAP performance for scientific computing*. In The 11-th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02), Jul 2002.
[11] *Introduction to DES*. http://www.rsasecurity.com/rsalabs/node.asp?id=2226

Appendix: Token Types of VTD

There are currently 15 token types defined in the VTD spec. To facilitate name space lookup, we define both attribute name type and name space type. Entity references are also given special attention—we define two attribute value types: one with entity reference; one without. The same thing also applies to text token.

Type value	***Type Name***	***Example***	***Variable Name***
0	*Start Tag*	<**example**>	**TOKEN_STARTING_TAG**
1	*End Tag*	</**example**>	**TOKEN_ENDING_TAG**
2	*Attribute Name*	<example **attrName1**="this">	**TOKEN_ATTR_NAME**
3	*Name Space*	<ns:example **xmlns:ns**="google">	**TOKEN_ATTR_NS**
4	*Attribute Value with entity reference*	<example property2="**that**">	**TOKEN_ATTR_VAL_NO_ENTITY**
5	*Attribute Value with entity reference*	<example property2="**th&gt;a t**">	**TOKEN_ATTR_VAL_HAS_ENTITY**
6	*Text without entity reference*	<example>**tasty fruit** </example>	**TOKEN_TEXT_NO_ENTITY**
7	*Text with entity reference*	<example>**tasty&gt;fruit**</exampl e>	**TOKEN_TEXT_HAS_ENTITY**
8	*Comment*	<!-- **this is a comment** -->	**TOKEN_COMMENT**
9	*Processing Instruction name*	<?**target** street="mission" ?>	**TOKEN_PI_NAME**
10	*Processing Instruction value*	<?target **street="mission"** ?>	**TOKEN_PI_VAL**
11	*XML declaration property name*	<?xml **version**="1.0" ?>	**TOKEN_DEC_ATTR_NAME**
12	*XML declaration property value*	<?xml version="**1.0**" ?>	**TOKEN_DEC_ATTR_VAL**
13	*CDATA value*	<![CDATA[**…data content…**]]>	**TOKEN_CDATA_VAL**
14	*DOCTYPE*	<!DOCTYPE **…**>	**TOKEN_DTD_VAL**

Aspect-Oriented Web Service Composition with AO4BPEL

Anis Charfi* and Mira Mezini

Software Technology Group
Darmstadt University of Technology
{charfi,mezini}@informatik.tu-darmstadt.de

Abstract. Web services have become a universal technology for integration of distributed and heterogeneous applications over the Internet. Many recent proposals such as the *Business Process Modeling Language (BPML)* and the *Business Process Execution Language for Web Services (BPEL4WS)* focus on combining existing web services into more sophisticated web services. However, these standards exhibit some limitations regarding modularity and flexibility. In this paper, we advocate an *aspect-oriented* approach to web service composition and present AO4BPEL, an aspect-oriented extension to BPEL4WS. With aspects, we capture web service composition in a modular way and the composition becomes more open for dynamic change.

Keywords: Adaptive web service composition, aspect-oriented programming, separation of concerns, BPEL.

1 Introduction

When the Web first emerged, it was mainly an environment for publishing data. Currently, it is evolving into a service-oriented environment for providing and accessing not only static pages but also distributed services. The Web Services [1] framework embodies the paradigm of Service-Oriented Computing (SOC) [2]. In this model, applications from different providers are offered as services that can be used, composed and coordinated in a loosely-coupled manner.

Web services are distributed autonomous applications that can be discovered, bound and interactively accessed over the Web. Although there can be some value in accessing a single web service, the greater value is derived from assembling web services into more powerful applications. Web service composition does not involve the physical integration of all components: The basic components that participate in the composition remain separated from the composite web service. As in *Enterprise Application Integration (EAI)*, specifying the composition of web services means specifying which operations need to be invoked, in what order, and how to handle exceptional situations [1].

* Supported by the German National Science Foundation (DFG) as part of the PhD program "Enabling Technologies for Electronic Commerce" at Darmstadt University of Technology.

Several composition languages have been proposed e.g., WSCI [3], BPML [4] and BPEL4WS [5]. These languages are process-based and have their origins in *Workflow Management Systems* (WFMS) [6]. A process defines the logical dependencies between the web services to be composed by specifying the order of interactions (control flow) and rules for data transfer between the invocations (data flow). In this paper, we identify two major problems of current composition languages.

The first problem concerns the modularity of the composition specification. A real-life composite web service usually offers several composite operations, each of which is specified as a business process that in turn aggregates other more elementary operations. Such hierarchical modularization of the composition specification according to the aggregation relationships between the involved business processes might not be the most appropriate modularization schema for some aspects of the composition that address issues such as *classes of service, exception handling, access control, authentication, business rules, auditing, etc*. The code pertaining to these concerns often does not fit well into the process-oriented modular structure of a web service composition, but rather cuts across the process boundaries. Without support for modularizing such *crosscutting concerns* [7] (this term is used to describe concerns whose implementation cuts across a given modular structure of the software) their specification is scattered around the processes and tangled with the specification of other concerns within a single process. This makes the maintenance and evolution of the compositions more difficult. When a change at the composition level is needed, several places are affected – an expensive and error-prone process.

The second problem that we identify with process-oriented composition languages concerns support for dynamic adaptation of the composition logic. Such languages assume that the composition logic is predefined and static, an assumption that does not hold in the highly dynamic context of web services. In fact, new web services are offered and others disappear quite often. In addition, the organizations involved in a web service composition may change their business rules, partners, and collaboration conditions. This motivates the need for more flexible web service composition languages, which supports the dynamic adaptation of the composition.

In order to tackle these limitations, we propose to extend process-oriented composition languages with aspect-oriented modularity mechanisms. The aspect-oriented programming (AOP for short) paradigm [9] provides language mechanisms for improving the modularity of *crosscutting concerns*. Canonical examples of such concerns are authorization and authentication, business rules, profiling, object protocols, etc. [10]. The hypothesis underlying AOP is that modularity mechanisms so far support the hierarchical decomposition of software according to a single criterion, based e.g., on the structure of data (a.k.a. object-based decomposition) or on the functionality to be provided (a.k.a. functional decomposition). Crosscutting modularity mechanisms [7] supported by AOP aim at breaking with this tyranny of a single decomposition [11] and support modular implementation of crosscutting concerns. Furthermore, with support for *dynamic weaving* [12, 13, 14], aspects can be activated/deactivated at runtime. In this way, aspects can also be used to adapt the application's behavior dynamically.

In this paper we present an aspect-oriented extension to BPEL4WS (BPEL for short) and show how this extension is useful for both improving the modularity of web service composition specifications and supporting dynamic adaptations of such compositions. Our approach is not specific to BPEL, though, and can be applied to any process-oriented composition language that supports executable business processes. BPEL was chosen as the basis technology merely because it is becoming the standard language for web service composition.

The remainder of the paper is organized as follows. Sec. 2 gives a short overview of how web service compositions are expressed in BPEL and discusses the limitations of this approach. Sec. 3 gives an overview of our aspect-oriented extension to BPEL and discusses how the limitations identified in Sec. 2 are addressed by it. We report on related work in section 4. Sec. 5 concludes the paper.

2 Process-Based Web Service Composition

In this section, we shortly introduce web service composition with BPEL as a representative for process-oriented web service composition languages and then consider its limitations.

2.1 Introduction to BPEL4WS

BPEL is a workflow-based composition language. In traditional workflow management systems, a workflow model represents a business process that consists of a set of basic and structured activities and the order of execution between them [15]. BPEL introduces control structures such as loops, conditional branches, synchronous and asynchronous communications. The building blocks of business processes are activities. There are *primitive* activities such as <invoke> and *structured* activities that manage the overall process flow and the order of the primitive activities. *Variables* and *partners* are other important elements of BPEL. Variables are used for data exchange between activities and partners represent the parties that interact with the process. Executable BPEL processes can run on any BPEL-compliant execution engine such as BPWS4J [16]. The execution engine orchestrates the invocations of the partner web services according to the process specification.

For illustration, Listing 1 shows a simple BPEL process from [16]. This process returns a string parameter back to the client whenever the operation *echo* is called. It consists of a <*sequence*> activity that contains two basic activities. The activity <*receive*> specifies that the process must wait until the client calls the operation *echo*. The activity <reply> specifies that the process has to send the message contained in the variable *request* to the client.

```
<process name = "echoString" …/>
 <variables>
   <variable name="request" messageType="StringMessageType"/>
 </variables>
 <partners>
   <partner name="caller" serviceLinkType="tns:echoSLT"/>
 </partners>
 <sequence name="EchoSequence">
   <receive  partner="caller" portType="tns:echoPT"
             operation="echo" variable="request"
             createInstance="yes" …/>
   <reply    partner="caller" portType="tns:echoPT"
             operation="echo"    variable="request"
             name="EchoReply"/>
 </sequence>
</process>
```

Listing 1. A simple process in BPEL4WS

2.2 Limitations of BPEL4WS

BPEL exhibits two major shortcomings: (a) lack of modularity for modeling crosscutting concerns and (b) inadequate support for changing the composition at runtime. In the following we elaborate on each of them.

2.2.1 Lack of Modularity in Modeling Crosscutting Concerns. As already mentioned in the introduction, a hierarchical modularization of the web service composition according to the aggregation relationships between the involved business processes might not be the most appropriate modularization schema for aspects of the composition that address issues such as classes of service, exception handling, access control, authentication, business rules, auditing, etc. Let us illustrate by the example of a simple travel service shown in Figure 1 how the code pertaining to these concerns cuts across the process boundaries and is not modularized. Our example travel web service provides the operations *getFlight* and *getHotel* which are specified as BPEL business processes (schematically represented by the vertical bars in Figure 1). A production web service usually provides several composite operations targeting different market segments.

Now, let us consider auditing [17] - an essential part of any workflow management system concerned with monitoring response times, logging, etc. An organization that composes external partner services is interested in measuring the response times of its partners because the response time of its service depends on those of the partners. The code needed for performing various auditing tasks will be scattered around the processes for composite operations, tangled in all these places with other concerns pertaining to these operations. This is because BPEL does not provide means to express in a modular way (in a dedicated module) at which points during the execution of various processes of a composite web service to gather, what auditing information. The resulting composition definition becomes complex and difficult to reason about it.

Similar modularity deficiencies can also be observed if we consider the calculation of the price for using the composite travel service in Figure 1. The code for the price calculation can certainly be encapsulated in some externalized web service. Both operations *getFlight* and *getHotel* in Figure 1 do indeed share such a common billing web service, *ws1*. What is not encapsulated, though, is the decision about “where” and “when” to trigger the billing functionality, i.e., the protocol that needs to be established between the operations *getFlight* and *getHotel* and the billing service. The code that is responsible for invoking the billing service is not modularized in one place: It crosscuts the modular process-based structure of the composition, as illustrated by the grey area cutting across the boundaries of the vertical bars (processes) in Figure 1. As a consequence, if the billing service *ws1* is replaced by some other billing service *ws2*, or the billing policy changes, we would have to change both process specifications for *getFlight* and *getHotel*. The problem is much more critical if we have more than two composite operations, which is to be expected in real complex web services. In general, one has to find out all the code that pertains to a certain concern and change it consistently. This is unfortunate especially because business rules governing pricing policies can be expected to change often.

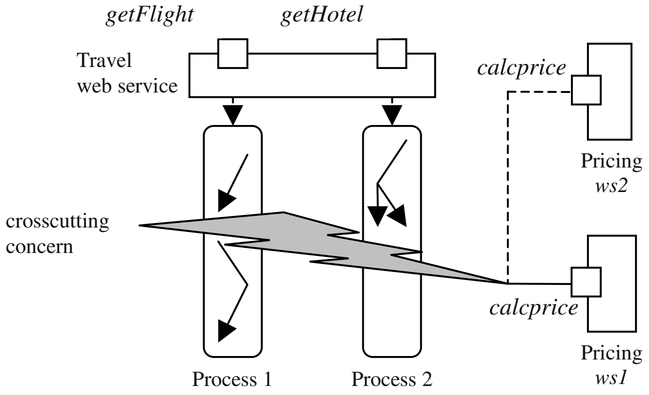


Fig. 1. Crosscuts in process-based web service composition

In general, business rules are typical examples of crosscutting concerns [18] in web service compositions. In the very competitive business context worldwide, business rules evolve very often (new partners, strategies…). Currently business rules are not well modularized in BPEL process specifications. Thus, when new business rules are defined, they get scattered over several processes. The resulting application is badly modularized, which hampers maintenance and reusability [18].

Again, the problem is that implementing business rules effects, in general, sets of points in the execution of the web services which transcend process boundaries. At present, BPEL does not provide any concepts for crosscutting modularity. This leads to tangled and scattered process definitions: One process addresses several concerns and the implementation of a single concern appears in many places in the process definition. If crosscutting concerns were well separated, process designers could concentrate on the core logic of process composition and process definitions become simpler. One would be able e.g., to exchange security policies without modifying the functional part (*independent extensibility*).

2.2.2 Changing the Composition at Runtime. When a BPEL process is deployed, the WSDL files of all the services participating in the composition must be known and once a process has been deployed, there is no way to change it dynamically. The only flexibility in BPEL is *dynamic partner binding*. This static view of the world is inherited from traditional workflow management systems from which the process-oriented web service composition model emerged; WFMS exhibit a major deficiency, namely the inadequate support of *evolutionary* and *on-the-fly* changes demanded by practical situations [19].

However, web service compositions implement cross-organizational collaborations, a context in which several factors call for *evolution* of the composition, such as changes in the environment, technical advances like updating of web services, addition or removal of partner web services, and variation of non-functional requirements. Some of these changes require dynamic adaptation, i.e., the composition must be open for dynamic modification, which is not possible in BPEL.

One might argue that if a runtime process change is required, we just have to stop the running process, modify the composition, and restart. This is actually the only way to implement such changes in BPEL. However, this is not always a feasible solution for several reasons. First, stopping a web service may entail a loss of customers. Second, especially in B2B scenarios, a composite operation may run very long, which may prevent us from stopping it. If we stop such a long-running transaction, we must roll back or compensate all previously performed activities. Third, the modification of the composition schema on a case-by-case basis leads to all kinds of exceptions. Last but not least, several situations require runtime ad-hoc derivation from the original planned business process. In general, such derivations are necessary if users are involved in making decisions and also if unpredictable events occur [19]. Such changes should affects only some process instances at runtime.

To illustrate the issues, consider a travel agency portal which aggregates information from partner web services (airline companies and hotel chains). The resulting travel portal is a composite web service exposing three composite operations as shown in Figure 2. Each of these operations is specified as a business process in BPEL e.g., the operation *getTravelPackage* aggregates two airline web services and a hotel web service.

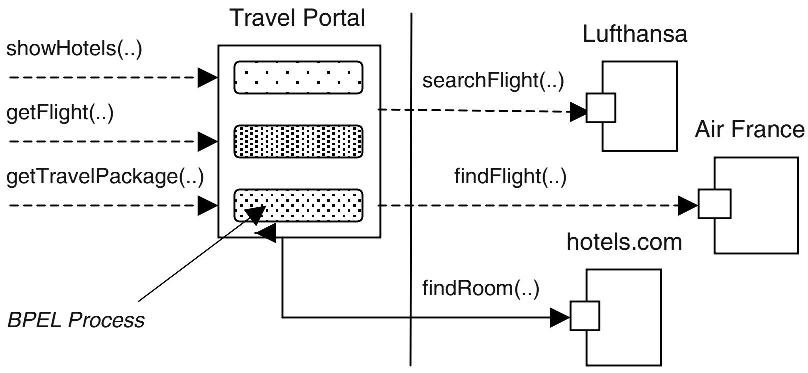


Fig. 2. A Travel portal as a composite web service

Assume that we want to add car rental to the composite operations *getTravelPackage* and *getFlight*. This way, when the client requests a flight or a travel package, she also gets propositions for car rental. One can also envisage variations of this adaptation, where such an offer is only made to frequent customers or to those clients that have specified interest in the offer in their profiles. In this case, the adaptation should be effective only for specific business process instances.

It should be noted that the adaptations of the composition logic such as the one illustrated here require, in general, a semantic adaptation mechanism to bridge the *compositional mismatch* [15]. In *component composition languages* [20] this kind of adaptation is enabled by *glue code*. Similarly, when we compose web services, we have to adapt their interfaces by writing glue code. BPEL supports only *data adaptability* by means of the <*assign*> activity. This activity allows one to create new messages or modify existing ones, using parts of other messages, XPath expressions, or literal values. Data adaptability deals with the structure of data. Glue code support is still missing in BPEL.

3 Aspect Oriented Web Service Composition

Aspect-Oriented Programming (AOP) [9] is a programming paradigm explicitly addressing the modularization of crosscutting concerns, which makes AOP the technology of choice to solve the problems discussed in Sec. 2. While it has been mostly applied to object-oriented programming, it is applicable to other programming styles [21], including the process-oriented style. We propose to use aspects as a complementary mechanism to process-oriented web service composition and argue that the definition of dynamic aspects at the BPEL level allows for more modularity and adaptability.

3.1 Introduction to Aspect-Oriented Programming

AOP introduces a new unit of modularity called *aspect* aimed at modularizing crosscutting concerns in complex systems. In this paper, we will use the terminology of AspectJ [22], the most mature AOP language today. In this terminology, there are three key concepts of AOP: *join points*, *pointcuts* and *advice*. Join points are points in the execution of a program [22]. In object-oriented programs, examples of join points are method calls, constructor calls, field read/write, etc.

In order to modularize crosscuts, a means is needed to identify related join points. For this purpose, the notion of a *pointcut* is introduced – a predicate on attributes of join points. One can select related method execution points, e.g., by the type of their parameters or return values, by pattern matching on their names, by their modifiers, etc. Similar mechanisms are available to select sets of related setter / getter execution points, sets of constructor calls / executions, exception handlers, etc. Current AOP languages come with predefined pointcut constructs (pointcut designators) in AspectJ.

Finally, behavioral effect at join points identified by a pointcut is specified in an *advice*. The advice code is executed when a join point in the set identified by the pointcut is reached. It may be executed before, after, or instead, the join point at hand, corresponding to *before*, *after* and *around* advice. The code specified in a before, respectively after advice is executed before, respectively after the join points in the associated pointcut have executed. With the around advice the aspect can control the execution of the original join point: It can integrate the further execution of the intercepted join point in the middle of some other code to be executed around it.

An aspect module consists in general, of several pointcut definitions and advice associated to them. In addition, it may define state and methods which in turn can be used in the advice code. Listing 2 shows a simple logging aspect in *AspectJ*, which defines a pointcut *loggableMethods* specifying **where** the logging concern should be integrated into the execution of the base functionality – in this case, the interesting join points are the executions (the *call* pointcut designator) of all public methods called *bar*, independent of the class they are defined in, their return type, as well as the number and type of the parameters (wildcards * and ..). The aspect also defines an advice associated to the pointcut *loggableMethods* that prints out a logging message *before* any of the points in *loggableMethods* is executed. The advice specifies **when** and **what** behavior must execute at the selected join points.

```
            public aspect Logging{

Where ?  ▶  pointcut loggableMethods(): call(public * *.bar(..)) ;

When ?   ▶    before() : loggableMethods()
              {
What ?   ▶          System.out.println("foo called" );
               }
            }
```

Listing 2. A logging aspect in AspectJ

Integrating aspects into the execution of the base functionality is called *weaving*. In static AOP approaches, as e.g., in AspectJ, at compile-time/load-time pointcuts are mapped to places in the program code whose execution might yield a join point at runtime. The latter are instrumented to add calls to advice and eventually dynamic checks that the identified places in code do actually yield a join point at runtime [23]. In dynamic AOP [12, 13, 14] languages, aspects can be (un)deployed at application runtime, the behavior of which can thus be adapted dynamically.

3.2 Overview of AO4BPEL

Here we present AO4BPEL, an aspect-oriented extension to BPEL4WS, in which aspects can be (un)plugged into the composition process at runtime. Since BPEL processes consist of a set of activities, join points in our model are well-defined points in the execution of the processes: Each BPEL activity is a possible join point. Pointcuts in AO4BPEL are a means for referring to (selecting) sets of join points that span several business processes at which crosscutting functionality should be executed. The attributes of a business process or of a certain activity can be used as predicates to choose relevant join points. E.g., to refer to all invocations of a partner web service, we use the attributes *partnerLink* and *portType* of the activity <invoke>. Since BPEL processes are XML documents, *XPath* [24] – a query language for XML documents – is a natural choice as the pointcut language. In an AO4BPEL aspect, the element <pointcut> is an XPath expression selecting those activities where the execution of additional crosscutting functionality will be integrated. XPath provides logical operators, which can be used to combine pointcuts.

Like AspectJ, we support *before*, *after* and *around* advice. An *advice* in AO4BPEL is an activity specified in BPEL that must be executed before, after or instead of another activity. The *around* advice allows replacing an activity by another (dummy) activity. Sometimes we need to define some advice logic which cannot be expressed in BPEL4WS. One could use code segments in a programming language like Java in Collaxa's JBPEL [25] for this purpose. However, this breaks the portability of BPEL processes, which is the reason for us to use what we call *infrastructural web services*. Such services provide access to the runtime of the orchestration engine. We set up a *Java code execution web service*, which invokes an external Java method in a similar way to Java Reflection. Each code snippet that is required within an AO4BPEL advice can be defined as a static method in a Java class.

Figure 3 sketches the overall architecture of our aspect-aware BPEL orchestration engine. The system consists of five subcomponents: the process definition and deployment tool, the BPEL runtime, the aspect definition and deployment tool, the aspect manager, and the infrastructural services. The core components are the *BPEL runtime* and the *aspect manager*. The *BPEL runtime* is an extended process interpreter. It manages process instances, message routing and takes aspects into account. The *aspect definition and deployment tool* manages the registration and activation of aspects. The *aspect manager* controls aspect execution.

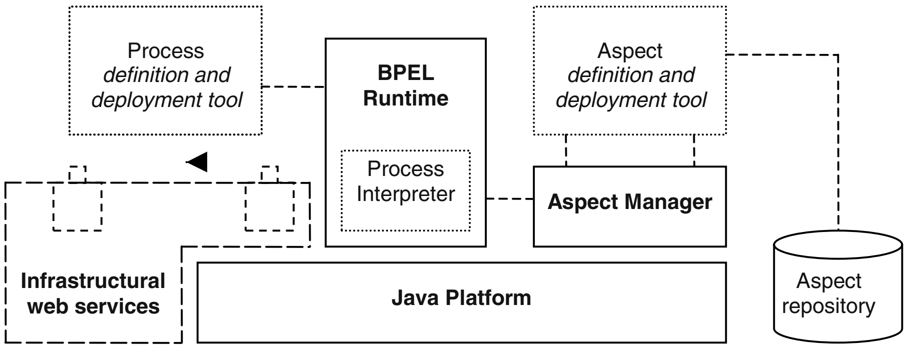

Process
definition and deployment tool
BPEL Runtime
Process Interpreter
Aspect
definition and deployment tool
Aspect Manager
Infrastructural web services
Java Platform
Aspect repository

Fig. 3. Architecture of an aspect-aware web service composition system

In our first implementation, we intend to support only <invoke> and <reply> join points because basic activities represent the interaction points of the composition with the external partners. The most straightforward way to implement a dynamic aspect-aware orchestration engine is to extend the process interpreter function to check if there is an aspect before and after the interpretation of each activity. If this is the case, the aspect manager executes the advice and then returns control to the process interpreter. We believe that for the first prototype, the performance overhead induced by these local checks is negligible compared to the cost of interacting with an external web service.

3.3 Examples Revisited

In this section, we show how the examples from Section 2 are modeled in AO4BPEL.

3.3.1 Modularizing Non-functional Concerns. To illustrate the modularization of crosscutting concerns, consider the AO4BPEL aspect *Counting* in Listing 3 which collects auditing data: It counts how many times the operation *searchFlight* of Lufthansa has been invoked. The counting advice must execute after each call to that operation. Ideally, we have to save this information into a file in order to evaluate it later, i.e., we need to access the file system. This cannot be done in BPEL and some programming logic is necessary. The *Java code execution web service* comes into play here. This web service provides the operation *invokeMethod* which takes as parameters the class name, the method name, and the method parameters as strings. It supports only primitive parameter types i.e., integers and strings. When the process execution comes to a join point captured by the pointcut *Lufthansa Invocations*, the static method *increaseCounter* is called, which opens a file, reads the number of invocations, increments it and then saves the file to disk.

```
<aspect name="Counting">
<partnerLinks>
  <partnerLink name="JavaExecWSLink" …/>
</partnerLinks>
<variables>
 <variable name="invokeMethodRequest" …/>
</variables>
<pointcutandadvice type="after">
 <pointcut name="Lufthansa Invocations">
   //process//invoke[@portType ="LufthansaPT" and
   @operation ="searchFlight"]
 </pointcut>
 <advice>
  <sequence>
   <assign>
    <copy>
     <from>increaseCounter</from>
     <to variable="invokeMethodRequest" part="methodName"/>
    </copy>…
   </assign>
   <invoke partnerLink="JavaExecWSLink" portType="JavaExecPT"
           operation="invokeMethod"
           inputVariable="invokeMethodRequest"/>
  </sequence>
 </advice>
</pointcutandadvice>
</aspect>
```

Listing 3. The counting aspect

This aspect shows how crosscutting concerns can be separated from the core of the composition logic. The monitoring functionality is not intertwined with the process definition. Moreover, we can define several monitoring aspects implementing different policies and weave the appropriate one according to the context.

3.3.2 Changing the Composition. In Section 2, we wanted to add car rental business logic into the composite operations *getTravelPackage* and *getFlight,* and argued that such an adaptation cannot be performed dynamically with BPEL. For achieving the same goal in AO4BPEL, the administrator defines the aspect *AddCarRental* shown in Listing 4. This aspect declares a pointcut, which captures the accommodation procurement activity in the *getTravelPackage* and the flight procurement activity in *getFlight.* The car rental activity must be executed after the join point activities referred to by the pointcut (after advice). This aspect also declares partner links, variables, and assignment activities. The <*assign*> activity is required to transform the data returned by the operations *getTravelPackage* and *getFlight.*

```
<aspect name="AddCarRental">
<partnerLinks>
 <partnerLink name="carRentalPortal" …/>
</partnerLinks>
<variables>
 <variable name="getCarRequest" …/>
 <variable name="getCarResponse" …/>
</variables>
<pointcutandadvice type= "after">
<pointcut name="accommodation procurement">
 //process[@name="getTravelPrcs"]//sequence[@name="FlightHotel"]
 //invoke[@portType ="HotelPT" and @operation ="findRoom"] or
 //process[@name="getFlightPrcs"]//flow[@name="FlightSearchFlow"]
</pointcut>
<advice>
 <invoke partnerLink="CarRentPortal" portType="carRentPT"
         operation="getCar" inputVariable="getCarRequest"
         ouputVariable="getCarResponse"/>
 <assign>
  …
 </assign>
</advice>
</pointcutandadvice>
</aspect>
```

Listing 4. The car rental aspect

The administrator must register the aspect with the BPEL execution engine. During the registration, the *aspect definition and deployment unit* of our BPEL engine requests the programmer to input the WSDL and the port address of the *car rental* web service. This step also requires the *partnerLinkTypes* that are used by the aspect. The aspect becomes active only after explicit deployment. The aspect activation can be performed dynamically while the respective process is running. This way, we apply the adaptation behavior at runtime.

One can conclude that this aspect tackles the problem outlined in Section 2. It allows for dynamic change. We specified the new business rule in a modular way as an aspect. If business rules change, we only have to activate/deactivate the appropriate aspect at execution time.

4 Related Work

Several research works recognize the importance of flexible and adaptive composition. *Self-Serv* [26] is a framework for dynamic and peer-to-peer provisioning of web services. In this approach, web service composition is specified declaratively by means of state charts. *Self-Serv* adopts a distributed decentralized, peer-to-peer orchestration model, whereby the responsibility of coordinating the execution of a composite service is distributed across several *coordinators*. This orchestration model provides greater scalability than a centralized one w.r.t. performance. In addition, Self-Serv introduces the concept of *service communities*, which are containers for alternative services. Separating the service description from the actual service provider increases the flexibility. However, unlike our approach, Self-Serv does not support dynamic process changes such as adding new web service type to the composition (cf. Sec.2).

Örriens et a.l [27], present a framework for *business rule driven* composition providing composition elements like *activities* and *composition rules*. Due to the separation of the activities from the specification of their composition, the latter can easily evolve by applying new composition rules. With AO4BPEL aspects, we achieve a similar effect of separating the main activities from their composition logic, while remaining compliant with BPEL4WS - a de-facto composition standard. In contrast to AO4BPEL, the approach presented in [27] does not support dynamic change and is rather geared towards the service composition lifecycle.

Adaptive workflow [19, 28, 29] provides flexible workflow models that are open for change. Similar to these works, AO4BPEL aims at making process-based composition more open for change. In addition, it addresses the issue of process modularity. We think that many concepts and results from adaptive workflow remain valid for our work e.g., the *verification and correctness of workflow modifications*.

Casati et. al, [29] present *eFlow*, which is a platform for specifying, enacting, and monitoring composite e-services (a predecessor of web services). eFlow models composite web services using graphs. It supports dynamic process modifications and differentiates *ad hoc change* (applies to a single process instance) and *bulk* change (applies to many or all process instances). Unlike AO4BPEL, eFlow supports change by migration of process instances from a *source schema* to a *destination schema*. This migration obeys several *consistency rules*. The advantage of our approach over eFlow is that we do not have to migrate the whole process instance from a source schema to a destination schema, we just weave sub-processes or advices at certain join points.

Dynamic AOP [12, 13, 14, 30] has been recognized as a powerful technique for dynamic program adaptation also by other authors. In [31], dynamic aspects are used for third-party service integration and for hot fixes in mobile communication systems. The idea of using AOP for increasing the flexibility of workflows was identified in [32], where the authors propose to model workflow according to different decomposition perspectives (dataflow, control flow, and resources). Along the same lines, *Bachmendo and Unland* [33] propose using dynamic aspects for workflow evolution within an object-oriented workflow management system.

There are two important differences between the approaches cited in the foregoing paragraph and the work presented here. First, none of the works presented in [31, 32, 33] directly targets the domain of web services. Second, all these approaches (mis)use dynamic AOP merely as an adaptation (patch) technique and do not use it as a modularization technique for crosscutting concerns, which it actually is. The second argument also applies to [34] – the only work that intends to apply dynamic AOP to web service composition known to the authors of this paper. In [34], dynamic aspects are used to hot-fix workflow and to configure and customize the orchestration engine. This approach is more *interceptor-based* than really aspect-oriented. Similar to [32, 33], the proposal in [34] uses dynamic AOP merely as an adaptation technique and not as a modularization technique for crosscutting. The advantage of our approach over these works is the *quantification* [35] i.e., the pointcut language of AO4BPEL which allows us to capture join points that span several processes in a modular and abstract way. We illustrated the crosscutting nature of functional and non-functional concerns especially observable when several processes are considered.

WSML [36] is a client-side web service management layer, which realizes dynamic selection and integration of services. WSML is implemented using the dynamic AOP language JAsCo [37]. Our work on AO4BPEL and the work on WSML nicely complement each other, since WSML does not address the problem of web service composition, while AO4BPEL is not concerned with the middleware issues of web-service management.

5 Conclusion

In this paper, we discussed limitations of web-service composition languages with respect to modularity and dynamic adaptability. We argued that the overall static model of the web-service composition languages inherited from the workflow management systems is not able to address issues resulting from the highly dynamic nature of the world of web services. To address these limitations, we presented an aspect-oriented extension to BPEL4WS where aspects and processes are specified in XML. We discussed how our approach improves process modularity and increases the flexibility and adaptability of web service composition.

The major contribution of our work is to elucidate why AOP is the right technique to address the problems of BPEL by arguing that these problems emerge due to the lack of crosscutting modularity. BPEL processes are the counterpart to classes in OOP. That is, similar to OO which mainly supports hierarchical decomposition in objects that are composed of other simpler objects the process-oriented languages support hierarchical decomposition of systems into processes that are recursively composed of other processes. Such a single decomposition schema while appropriate for expressing the process structure in a modular way, is, however, not well-suited for expressing other concerns that cut across module boundaries. With its ability to quantify over given module boundaries [35] by means of pointcuts that span several processes, AO4BPEL is able to capture such concerns in a modular way. This is actually the key message put forward in this paper.

Currently, we are still working on a first prototype of the aspect-aware orchestration engine, which manages BPEL aspects and processes. In the future, we intend to enhance the interface for context passing between BPEL aspects and the base processes. We will also examine the correctness properties of process adaptation. Another direction for future research is to investigate whether semantic web and ontologies may enable semi-automatic generation of adaptation aspects in case of dynamic changes.

Acknowledgements. We would like to thank D. Karastoyanova, M. Haupt, and S. Kloppenburg for comments on earlier drafts of this paper. We also thank the anonymous reviewers for their suggestions for improving this paper.

References

1. G. Alonso, F. Casati, H. Kuno,V. Machiraju. *Web Services: Concepts, Architectures, and Applications*. Springer, 2004.
2. M. P. Papazoglou. *Service-Oriented Computing: Concepts, Characteristics and Directions.* 4th Int. Conference on Web Information Systems Engineering (WISE'03), Italy, 2003.
3. A. Arkin et al., *Web Service Choreography Interface* 1.0, W3C, 2002.
4. A. Arkin et al., *Business Process Modeling Language*- BPML 1.0, 2002.
5. T. Andrews et al., *Business Process Execution Language for Web Services* 1.1, May 2003.
6. D. Georgakopoulos, M. Hornick, A. Sheth. *An Overview of Workflow Management: from process modeling to workflow automation infrastructure.* Distributed and Parallel Databases, April 1995.
7. H. Masuhara, G. Kiczales. *Modeling Crosscutting in Aspect-Oriented Mechanisms.* In
8. Proceedings of ECOOP2003, LNCS 2743, pp.2-28, Darmstadt, Germany, 2003.
9. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin. *Aspect-oriented Programming*. In ECOOP'97, LNCS 1241, pp. 220-242, 1997.
10. R. Laddad. *AspectJ in Action*. Manning Publications, 2003.
11. P. Tarr, H. Ossher, W. Harrison, S.M. Sutton. *N degrees of Separation: Multidimensional separation of concerns.* Proc. ICSE 99, pp. 107-119, 1999.
12. C. Bockisch, M. Haupt, M. Mezini, K. Ostermann. *Virtual Machine Support for Dynamic Join points.* Proceedings of the 3rd AOSD conference, Lancaster, UK, 2004.
13. R. Pawlak, L. Seinturier, L. Duchien, G. Florin. *JAC: A Flexible Solution for Aspect-Oriented Programming in Java.* Proceedings of the 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, Japan, 2001.
14. B. Burke, M. Flury. *JBoss AOP*, http://www.jboss.org/developers/projects/jboss/aop.jsp.
15. R. Khalaf, N. Mukhi, S. Weerawarana. *Service-Oriented Composition in BPEL4WS.* WWW2003 conference, Budapest, Hungary, 2003.
16. The IBM BPEL4WS Java™ Run Time, http://www.alphaworks.ibm.com/tech/bpws4j.
17. V. Tosic, W. Ma, B. Pagurek, B. Esfandiari . *Web Services Offerings Infrastructure(WSOI) - A Management Infrastructure for XML Web Services*. Proc. of NOMS 2004, Seoul, 2004.
18. M. D'Hondt, V. Jonckers. *Hybrid Aspects for Weaving Object-Oriented Functionality and Rule-Based Knowledge*. Proceedings of the 3rd AOSD conference, Lancaster, UK, 2004.
19. Y. Han, A. Sheth, C. Bussler. *A Taxonomy of Adaptive Workflow Management.* CSCW'98 Workshop on Adaptive Workflow, USA, 1998.
20. F. Achermann, O. Nierstrasz. *Applications = Components + Scripts — A Tour of Piccola.* Software Architectures and Component Technology, Kluwer, 2001.
21. Y. Coady, G. Kiczales. *AspectC*, http://www.cs.ubc.ca/labs/spl/projects/aspectc.html.
22. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, W. Griswold. *An overview of AspectJ*. In Proceedings of the ECOOP 2001, Budapest, Hungary, 2001.
23. E. Hilsdale, J. Hugunin. *Advice Weaving in AspectJ*. Proceedings of the 3rd AOSD conference, Lancaster, UK, 2004.
24. J. Clark. *XML path language* (XPATH), 1999 http://www.w3.org/TR/xpath.
25. Collaxa BPEL Server, http://www.collaxa.com

26. B. Benatallah, Q. Sheng, M. Dumas. *The Self-Serv Environment for Web Services Composition.* IEEE Internet Computing, January / February 2003.
27. B. Orriëns, J. Yang, M.P. Papazoglou. *A Framework for Business Rule Driven Web Service Composition*. ER (Workshops), Chicago, USA, 2003.
28. C. Bussler. *Adaptation in Workflow management.* Proceedings of the Fifth International Conference on the Software Process, CSOW, Illinois, USA, June 1998.
29. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M.Shan. *Adaptive and Dynamic Service Composition in eFlow.* In Proc. of the Int. Conference on Advanced Information Systems Engineering (CAiSE), Sweden, June 2000.
30. Y. Sato, S. Chiba, M. Tatsubori. *A Selective Just-in-Time Aspect Weaver.* Proceedings of the GPCE 03 conference, LNCS 2830, Erfurt, September 2003.
31. R. Hirschfeld, K. Kawamura. *Dynamic Service Adaptation.* 4th International Workshop on Distributed Auto-adaptive and Reconfigurable Systems, Tokyo, Japan, 2004.
32. R. Schmidt, U.Assmann. *Extending Aspect-Oriented-Programming in order to flexibly support Workflows.* AOP Workshop, ICSE 98, USA, 1998.
33. B. Bachmendo, R. Unland. *Aspect-based Workflow Evolution.* Workshop on AOP and separation of concerns, Lancaster, UK, 2001.
34. C. Courbis, A. Finkelstein. *Towards an Aspect-Weaving BPEL-engine.* ACP4IS Workshop, 3rd AOSD conference, Lancaster, UK, 2004.
35. R.E. Filman, D.P. Friedman. *Aspect-Oriented Programming is Quantification and Obliviousness*. Advanced Separation of Concerns Workshop, OOPSLA 2000, Minneapolis, USA, 2000.
36. B. Verheecke, M. Cibran. *AOP for Dynamic Configuration and Management of Web Services.* International Conference on Web Services Europe 2003, Erfurt, 2003.
37. D. Suvee, W. Vanderperren, V. Jonckers. *JAsCo: an aspect-oriented approach tailored for component based software development.* 2nd AOSD conference, Boston, USA, 2003.

Coupled Signature and Specification Matching for Automatic Service Binding*

Michael Klein[1] and Birgitta König-Ries[2]

[1] Institute for Program Structures and Data Organization, Universität Karlsruhe, D-76128 Karlsruhe, Germany kleinm@ipd.uni-karlsruhe.de
[2] Institut für Informatik, Technische Universität München, D-85748 Garching, Germany koenigri@in.tum.de

Abstract. Matching of semantic service descriptions is the key to automatic service discovery and binding. Existing approaches split the matchmaking process in two step: signature and specification matching. However, this leads to the problem that offers are not found although they are functionally suitable if their signature is not fitting the requested one. Therefore, in this paper, we propose a matching algorithm that does not use a separated and explicit signature matching step, but derives the necessary messages from the comparison of pre- and postconditions. As a result, the algorithm not only finds all functionally suitable services even if their signatures do not match, but also is able to derive the messages needed for an automatic invocation.

Keywords: Automatic Service Discovery/Invocation, Matching Semantic Service Descriptions, OWL-S

1 Introduction

An important vision of service oriented computing is to enable semantic, dynamic service binding, i.e. it should become possible to automatically choose *and* invoke service providers at runtime.

To achieve this, appropriate means to describe and match services are needed. The techniques developed by both the web services and the semantic web communities offer a suitable basis, but fall short of realizing the vision.

Maybe the major drawback of existing approaches to service description and matching is that they focus on the message flow of services. Effects and preconditions of services are either regarded separately from the message flow or, more frequently, not regarded at all. The matching process is thus performed in two steps:

– *Signature matching.* Checks whether the exchanged messages of the requested and offered service fit together. In OWL-S, e.g., this is done by comparing the input and output properties. Typically, it is analyzed whether

* This work is partially funded by the Deutsche Forschungsgemeinschaft (DFG) within SPP 1140.

the client provides at least the inputs needed by the service and whether vice versa the service produces at least all the outputs needed by the client.
– *Specification matching.* Checks whether the offered service provides the requested functionality. In OWL-S, e.g., this is done by comparing the state transition given in the precondition and effect properties.

An offer fits a request if their descriptions match in both steps. If this is the case, the service can directly be invoked as the message flows are essentially the same.

The big problem of this approach with respect to dynamic service binding is that the assumption that equality of message flow equals equality of functionality is wrong. Services with identical messages flows may offer completely different functionality; services with identical functionality may use different message flows to achieve this functionality. Thus, if matching is based on the comparison of message flows, on the one hand, services offering a functionality different from the one intended by the requestor may be invoked, on the other hand, appropriate services are overlooked, if their message flow does not match the request. The latter case is very common:

– The request demands for outputs that are not specified in the offer description because they are constant. For example, the requestor searches for a printing service that informs him about the location of his printout after service execution. For the offerer, however, the location is not an output of the service description as his service always prints on the printer in Room 335.
– The offerer demands for inputs that are not specified in the request as the requestor did not know that this value was necessary or omitted this input because it was constant. For example, it could be possible that the requestor did not know that the offered printing service needed the location of the printout or he wanted the printout always to be in Room 350, so he did not specify it as a input in his request.
– The offer demands for inputs that are specified as outputs in the request. For example, the offerer could request for the location of the printout, while the requestor wants this value as output.

In these cases, the signature matching fails and prevents the requestor from using a functionally suitable service.

In this paper, we present a novel matching approach that does not rely on an explicit signature matching step, but derives the necessary messages from the comparison of pre- and postcondition. The algorithm operates on state based service descriptions, which have been presented in [1]. Here precondition and effect are described by states with integrated variables which represent the message flow. With this approach we are able to find service providers that offer the desired functionality, even if their signature differs from the one specified in the request. At the same time, we are able to determine precisely which messages need to be sent for a successful service invocation. With this knowledge, it becomes possible to automatically generate these messages from the request and thus to fully automate service invocation.

The paper is structured as follows: In Section 2, we inspect existing approaches on matching semantic service descriptions. After that, in Section 3, we revisit our state oriented service description by showing which additional description elements are needed and by giving an example for a request and an offer description. Section 4 introduces our novel matching algorithm that operates on these descriptions and couples signature and specification matching. Finally, a conclusion is given in Section 5.

2 Related Work

As addressed in the introduction, existing approaches compare services mainly by explicitly matching their messages descriptions.

A prominent representative is the *Semantic Matchmaker* of the Software Agent Group at CMU developed by Paolucci et al. [2]. The algorithm operates on OWL-S descriptions and tests if the request can provide all required inputs and if the offer's output satisfies the requestor's demands. As an exact match of the types is very strict, the matching degree `exact` is weakened: An output r of a request is also matching 'exactly' to an offer's output o if r is at direct subclass of o. When comparing inputs, the condition is inverted: r has to be a direct superclass of o. The reasoning behind this is that a service will only use the superclass to describe its functionality if the output offers *all* types of the direct subclasses. Besides `exact`, there are two other matching degrees: `plugIn` and `subsumes`. They allow a greater deviation from the original type. In any case, the approach relies on the message flow only, which leads to different problems as we have seen. Moreover, in cases of a non-exact match, the offer cannot be used directly as the messages of request and offer are not compatible. In the *Mind Swap* project [3], a similar approach is pursued.

An extension of this approach is the matcher for *LARKS* [4,5] which was developed by Sycara et al. In LARKS, services are described by four functional parameters `input, output, inConstraints` and `outConstraints`. The algorithm uses up to five filters to match the description. The first three filters use techniques from information retrieval, the fourth filter is the Semantic Matchmaker described above, the fifth filter separately compares the pre- and postconditions. In summary, the approach suffers from the same problems as the Semantic Matchmaker as it directly compares the message descriptions, too.

Trastour et al. at *agents@HP Lab* have developed a matching algorithm for service descriptions based on RDF [6]. The descriptions rely on a common RDF schema for an `Advertisement`. Thus, the comparison is performed by a graph matching approach: (1) Two descriptions are similar, if the root elements are similar and (2) two elements a and b are similar if (a) a is a subclass of b and each property of a which is also property or subproperty in b points to a similar element. It is also possible to attach special matching code to the elements. Although the algorithms seems to be very useful, the problems stemming from explicitly matching message descriptions reside. However, we will use this algorithm as starting point for our matching process (see Section 4.2: 'The Basic Algorithm') and adopt it to state oriented service descriptions.

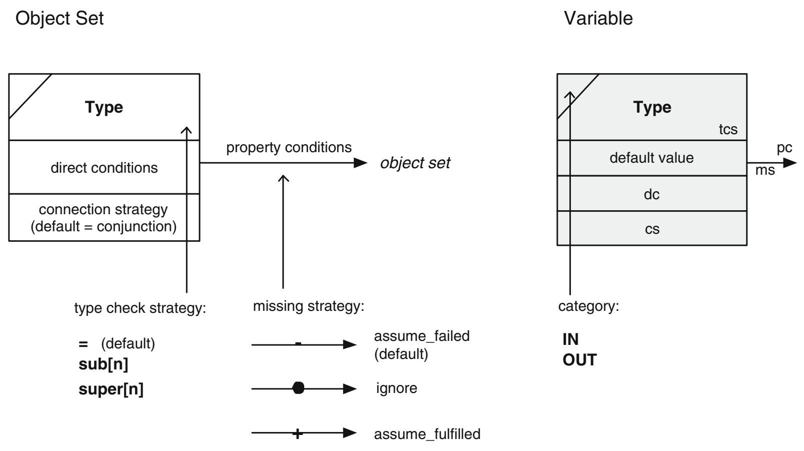


Fig. 1. How to define the enhanced description elements: object sets and variables.

Furthermore, approaches that rely on logical derivation (e.g. on reasoning in description logics) perform their operations on explicit message descriptions, like in [7], [8], [9], and [10].

To summarize, the major drawbacks of existing approaches are: Most of them rely on signature matching. These are not able to find services that offer equivalent functionality with a different interface. Approaches that take the desired functionality into account and restrict the conditions on signature matching are able to find appropriate services. However, they are not able to automatically invoke them, as they do not include mechanisms to map the message flows expected by requestor and provider to one another.

3 State Oriented Service Descriptions

The main idea of this paper is to avoid an explicit matching of message descriptions but to derive necessary messages by comparing pre- and postconditions from request and offer. This becomes possible if the description of input and output is integrated into the states of pre- and postcondition. Therefore, in a first step, we will revisit our *purely state oriented description*[1] in the Sections 3.2 and 3.3. The description is based on classic elements like classes, properties, and instances. However, it needs additional description elements: sets and variables. We will introduce them in the next section.

3.1 Additional Description Elements: Sets and Variables

To describe offers and requests in a state oriented manner, we need enhanced description elements: declarative object sets and variables. **Object sets** are sets

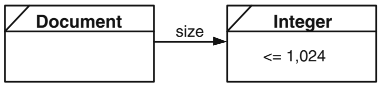
[1] A predecessor of this description was introduced in [1]

containing objects of one (or more) types. They should not be mixed up with classes which contain *all* objects of the corresponding type.

An object set is described declaratively, i.e. by giving conditions that have to be fulfilled in order to be a set member. Graphically, such a set is represented as a rectangle with a small triangle in the left upper corner (see Figure 1). The following three conditions are possible in a definition:

- A *type t*. All objects in the set have to be exactly of this type t. This restriction can be weakened by a different type check strategy (see below). We access the type of a set s by `type(s)`.
- A list of *direct conditions*. Direct conditions are simple conditions that are restricting the members of the set. Each object in the set has to obey all these conditions. If the type of the set is a primitive datatype (like Integer, String, Date etc.) all typical comparison operations for this type are allowed; if the type is a user defined type (like Person, Format etc.) only direct comparisons to named objects of this type are allowed (like '= $\underline{\text{pdf}}$'). We access the direct conditions of a set s by `directConditions(s)`
- In case of a complex type: a list of *property conditions* which are depicted as named arrows at the rectangular. Only properties of the corresponding type can be used as property conditions. For this paper, only properties with cardinality $< 0, 1 >$ or $< 1, 1 >$ are allowed. Each of these property conditions p points to another object set y and leads to the following restriction for the members of the set x: An object can only be member of set x if it has a defined property p pointing to an instance that is member of y. By default, all property conditions of the set are connected conjunctively. This can be changed by a different connection strategy (see below).

This example shows two object sets:



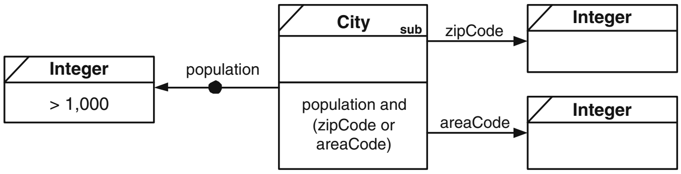
The set on the right hand side contains integer values that are smaller than 1024 (direct condition). The set on the left hand side contains Document objects that have a size which is in the other set (property condition).

The default behavior of a set can be changed by specifying different strategies:

- A *type check strategy (tcs)* defines the type the objects in the set have to have. By default, only objects of exactly the given type t are allowed in the set. This is expressed by the tcs '='. If also subtypes of t are allowed, the tcs **sub[n]** can be used where the optional parameter n is the maximum distance in the ontology. For example, **sub[1]** means that objects of type t and t's children are allowed. **super** accepts super types of t. We access the type check strategy of a set s by `tcs(s)`.
- The *connection strategy (cs)* changes the way in which the single results of the property conditions are connected. By default, they are connected conjunctively. The cs is specified as a boolean expressions where the operations `and`, `or`, and `not` are allowed. We access a single connection operation of the property condition p by `cs(p)`.

– The *missing strategy (ms)* specifies the behavior in case of a missing property at an object which is tested for set membership. If for example a property condition for a set of type Document specifies that the document's format has to be pdf or ps, but the current object has not specified that property, the missing strategy decides how to proceed with this object: By default, the missing strategy is *assume_failed*, which means that in case of a missing property the condition should be regarded as if it had failed. This strategy is depicted by a minus on the property condition arrow (or left blank as it is the default). More strategies are *ignore* depicted by a circle, which skips the condition if it is not defined in the object, and *assume_fulfilled* depicted by a plus, which means that the condition should be regarded as if it had succeeded. We access the missing strategy of a property condition *p* by `ms(p)`.

This example shows an object set of Cities:



The strategies change its default semantics: Also objects of subclasses of City are allowed in the set, each object has to have a defined post code *or* area code, and the population has to be larger than 1000 – but objects without a defined population are possible, too.

We have chosen this representation of object sets, because it is very intuitive as it clearly separates different aspects like the conditions themselves, the missing strategy and the connecting strategy. Also, it allows to easily extend or adapt one of the aspects independent of the others and is therefore very suitable for future extensions.

Variables are a special kind of object sets which are exclusively needed when describing services. They represent positions where the service offerer or requestor have to insert information before or after service execution. In any case, the information that is inserted has to be a member of the object set, i.e. it has to obey all given conditions. Therefore, variables have the range of the corresponding object set and possibly one single assigned value. Thus, they can be used as if they were an instance. Variables are depicted in the same way as object set, but they use a gray rectangle and have two additional characteristics (see Figure 1, right hand side):

– A *category*. It specifies by whom the information has to be inserted. IN means that the service requestor has to fill in the information, i.e. assign a single value from its range, OUT means that the service offerer will fill in the information.
– A *default value*. This value is inserted by default if the service offerer or requestor did not specify a value. Note that the default value has to be a member of the set, too.

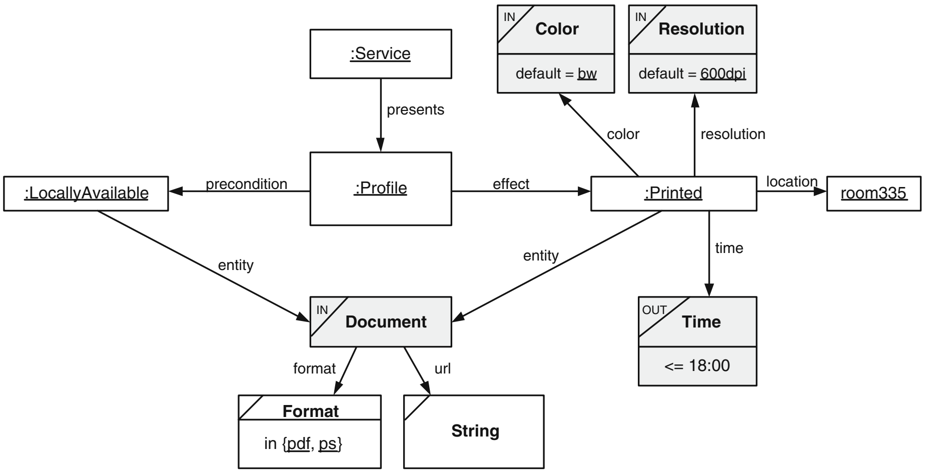


Fig. 2. Example of a state oriented service description of an **offered** printing service.

3.2 Offer Descriptions

The two additional description elements help to overcome the separation of message flow and state transition. Now, it is possible to remove the explicit description of the exchanged messages and integrate it with the help of variables into the states. This leads to state oriented service descriptions (see [1]).

In Figure 2, the offer description of the example printing service is shown as such a state based service description. The input and output properties are removed while the state descriptions are refined by using a common Document object. Thus, the described service transforms documents from the state LocallyAvailable into the state Printed. Moreover, the message flow is integrated as variables (depicted as gray rectangles) in the following manner:

– Values of *incoming messages* (former inputs) are integrated as IN variables of the appropriate type. In the example, inputs are the desired Color and Resolution of the printout as well as the document itself. It is useful to specify default values for variables, e.g. the default printout will be black-white in a resolution of 600dpi. The two property conditions of the Document variable specify that the service expects objects with an arbitrary but defined URL as String and with a format which is pdf or ps.
– Values of *outgoing messages* (former outputs) are integrated as OUT variables. They represent the set of objects that can be expected as result when invoking the service. For example, the Time variable specifies that the finishing time of the printout is returned, its direct condition says that it will have a value smaller than 18:00.

As a result, on the one hand, the IN and OUT variables implicitly define the incoming and outgoing messages while their positions within the states clearly expresses their influence on the functionality of the service. Direct and property conditions restrict the possible values.

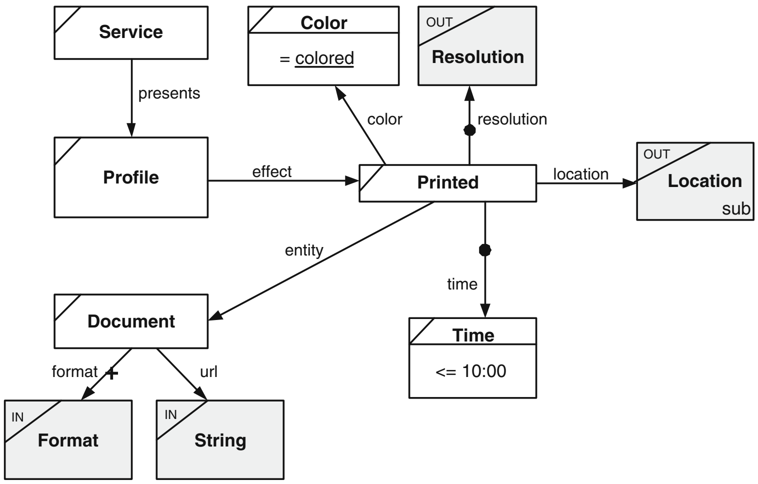


Fig. 3. Example of a state oriented, set based service description of a **request** for a printing service.

3.3 Request Descriptions

Request descriptions are defined in a similar way. The explicit specification of the message flow is omitted – instead, the desired interface is integrated into the states as variables like above. However, service requests have a different purpose than service offers, which should be reflected in their descriptions. In [11], we showed that a single instance graph is not suitable to describe the functionality desired by the requestor, because it is not possible to include the information what deviation from this perfect service is tolerated by the requestor. We proposed to use sets of objects in request descriptions instead. Thus, the request is an object set of type Service where elements of the sets are suitable services.

Figure 3 shows an example request description in a state oriented, set based form. The requestor asks for service objects that have a profile with an effect of type Printed. The printed entity should be the document with the url and format defined by the requestor in a concrete service call (IN variable). In any case the printout should be colored and finished before 10 o'clock (direct conditions in the sets of type Color and Time). After service execution the requestor wants to get informed about the resolution and location of the printout (OUT variables). Also subclasses of Location are allowed (type check strategy **sub**).

Note that request descriptions will typically not contain any conditions about the precondition, but only about the desired effects, as the requestor does not know what preconditions could be needed by a concrete offer. Thus, the checking whether the preconditions of a suitable offer can be fulfilled is done after the matching process and omitted in this paper.

4 Coupled Signature and Specification Matching

4.1 Introduction

The matching process has two tasks: On the one hand, it has to determine if a offer description fits a given request description, on the other hand, it has to assign concrete values to the variables in the descriptions which is necessary to derive the messages needed later. More precisely, the following variables need to be filled:

- All *IN variables of the offer*, which are called *OffIN* variables in the following. They are needed for generating the message which invokes the offered service.
- All *OUT variables of the request*, which are called *ReqOUT* variables in the following. They are needed for providing the desired return values to the service requestor. The requestor can use the missing strategy *assume_fulfilled* or *ignore* to specify that he is not needing these values necessarily.

The OUT variables of the Offer (*OffOUT*) don't need to be used in cases where the requestor is not interested in them. Also, the IN variables of the request (*ReqIN*) are unproblematic as they are already replaced with concrete values before the matching process starts. For example, if the service requestor wants to use a service `print("http://domain.de/file.pdf", pdf)`, the corresponding values are assigned to the ReqIN variables before a suitable service is searched.

In principle, the matching process has to check a set inclusion: is the offered service (described as single instance) element of the requested service (described as set of instances)? Or more precisely: What assignment of OffIN and ReqOUT variables is necessary so that the offer is included in the set defined by the request? As both descriptions are graphs stemming from similar ontological concepts (= classes), an obvious basic technique for comparing both descriptions is a graph matching approach. Beginning with the root element of type Service, the two descriptions are traversed synchronously and compared step by step.

We will present this matching algorithm in several stages. First, we will explain the basic algorithm that simply assumes default strategies and is not capable of dealing with variables in the descriptions (Section 4.2). The problems with variables are analyzed in more detail in Section 4.3. After introducing necessary set operation, we present the full algorithm in Section 4.4.

4.2 The Basic Algorithm

The basic algorithm in pseudo code is shown in Listing 1.1. Parts that are denoted in brackets contain code that (a) is not used in the basic version of the algorithm because it handles variables (Lines 5, 6, and 20) or (b) is kept very simple by only regarding the default strategies (Lines 3, 15, and 17).

```
 1  boolean match(r,o)
 2  {
 3      [TypeCheckStrategy]: if (type(r) != type(o)) return false;
 4
 5      [OffOUT]
 6      [OffIN]
 7
 8      if (o not fulfills directConditions(r)) return false;
 9
10      matches = true;
11
12      for each defined property condition p of r
13      {
14          if (o.p is defined) singleresult = match(r.p, o.p);
15          else [MissingStrategy]: singleresult = false;
16
17          [ConnectingStrategy]: matches = matches && singleresult;
18      }
19
20      [ReqOUT]
21
22      return matches;
23  }
```

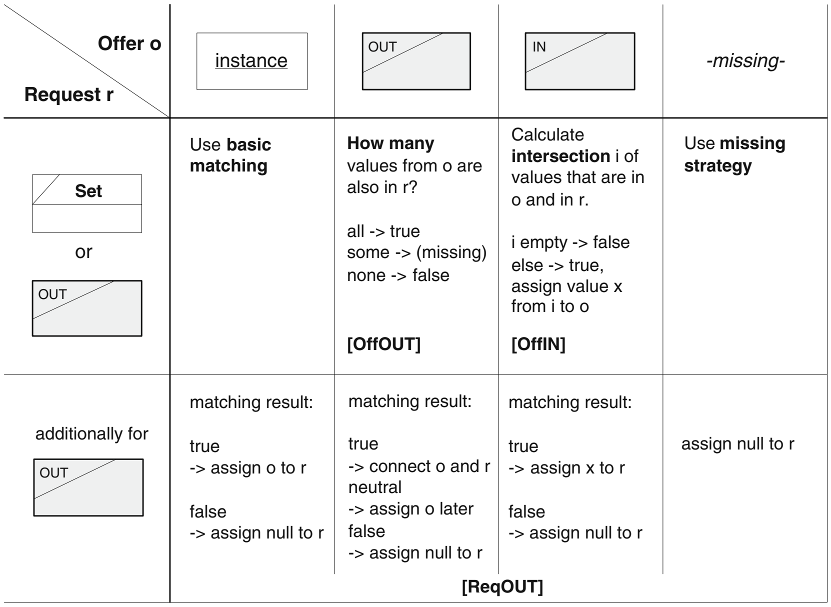
Listing 1.1. Basic matching algorithm

The algorithm expects two inputs: the root element r of a request description as well as the root element o of an offer description. If o is a semantically fitting offer for the request r, the method returns true, otherwise it returns false. Furthermore, the variables are filled in as a side effect by assigning concrete values to them. This is done in the hidden code segments and explained later.

The algorithm is recursive and operates in the following manner: At first, it checks in Line 3, if the two provided elements are of the same type. If not, the comparison fails and false is returned. In the full version, the algorithm has to perform this comparison according to the type check strategy of the set in request. In Line 8, it is checked whether o satisfies all direct conditions of r. If not, false is returned.

The loop beginning in Line 12 checks the property conditions of the set r. If the corresponding property is also defined in the offer o, the algorithm declines to the values $r.p$ and $o.p$ and compares them recursively (Line 14). The result of the single comparison is stored in `singleresult`. If p is not defined in o, in this basic version, a matching failure is assumed and false is assigned to `singleresult` (Line 15). In the full version, the algorithm has to handle missing values according to the missing strategy specified by the user.

The combination of the single results is done in Line 17. In this basis version, all partial results are combined conjunctively by default. Later, the user defined combination strategy is used. This provides the possibility to use disjunction or negation. After finishing the loop, the calculated result is returned.



Request r \ Offer o	instance	OUT	IN	-missing-
Set or OUT	Use **basic matching**	**How many** values from o are also in r? all -> true some -> (missing) none -> false **[OffOUT]**	Calculate **intersection** i of values that are in o and in r. i empty -> false else -> true, assign value x from i to o **[OffIN]**	Use **missing strategy**
additionally for OUT	matching result: true -> assign o to r false -> assign null to r	matching result: true -> connect o and r neutral -> assign o later false -> assign null to r	matching result: true -> assign x to r false -> assign null to r	assign null to r
	[ReqOUT]			

Fig. 4. Problems and solutions while matching variables.

Note that the matching process is driven by the structure of the request description, which should not contain any cycles.

4.3 Dealing with Variables

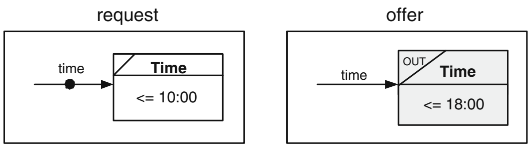
This basic version of the matching algorithm can only handle descriptions without variables. Variables are problematic for two reasons:

– Variables are undefined parts of the description which leads to problems when matching them.
– ReqOUT and OffIN variables have to be filled with concrete values. This is a prerequisite to allow for automatic service invocation.

The table in Figure 4 analyzes the problems in more detail and shows solutions for them. The elements which can appear in the request r are listed vertically; the elements of the offer o are listed horizontally. As the matching process is driven by the structure of the request description, only sets and OUT variables appear here. They can encounter single instances, IN and OUT variables, as well as missing values in the offer. As the ReqOUT variables are sets, too, the matching process is the same as with normal sets. Additionally, it has to be assigned a value to them. Thus, the table is split horizontally: the upper part is valid for sets and ReqOUT variables, the lower contains the additional parts for the ReqOUT variables.

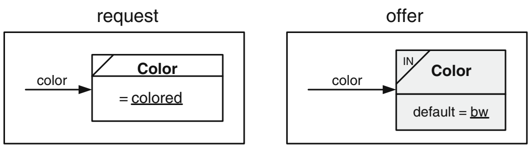
In the following, we will explain three interesting cases from the table which are marked with [OffOUT], [OffIN], and [ReqIN]. The algorithm can be improved by inserting the resulting code at the corresponding tag in the basic version. In the following o denotes the element from the offer, r the element from the request:

[OffOUT]. As the value of the OffOUT variable o is filled in by the offerer after a successful service execution, it has no assigned value during the matching process. However, as the OffOUT variable is also a set, the offerer has to take a value from this set. Therefore, the matcher has to check if the values of the set o could be elements of the requested set r. If all the values of o are also in r, the result of the matching result is definitely true, if none of the values of o is also in r, the matching result is definitely false. In other cases, i.e. if there are both values of o in and not in r, the matching result is undefined – the offerer could return a suitable value, but this is not guaranteed. In this case, the matcher acts conservatively and assumes the value of the OffOUT variable to be missing. Thus, the missing strategy decides the behavior.



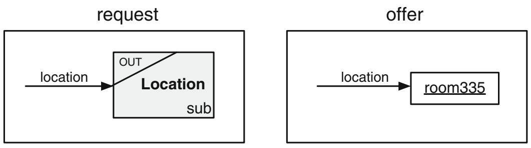
In our example, the Time set in the request is matched against the OUT variable of type Time in the offer. However, the matching result is undefined as the offerer guarantees a finishing time before 6pm while the requestor wants the service to be finished before 10am. Therefore, a missing value is assumed, where the missing strategy *ignore* (black circle on the arrow in the request) decides to skip the property condition completely.

[OffIN]. As the value of the OffIN variable can be assigned by the requestor himself, it has to be checked if there are values in r that could also be inserted into o. Thus, the intersection i of the sets o and r is calculated. If i is empty, the conditions of requestor and offerer are contradictory, and false is returned as the matching result. However, the default value of o is assigned to the OffIN variable as the service could be suitable anyway because of a non-standard connecting strategy. If i is not empty, one arbitrary value from i is selected and inserted into the OffIN variable o.



In the example, the Color set with the direct condition '= colored' is matched against the OffIN variable of type Color without conditions. The intersection contains the element colored which is assigned to the OffIN variable o.

[ReqOUT] If r additionally is a ReqOUT variable, its value is assigned according to the result of the matching process between o and r: In case of an unsuccessful matching, null is assigned to r. This is necessary because a negative matching result of r and o does not necessarily lead to an unsuccessful matching result of the whole descriptions since non-default connecting strategies are possible. In case of a successful matching result, if o has been an OffIN variable, the assigned value is also assigned to r, in case of an OffOUT variable, the variables are connected, i.e. after a successful service execution the result of the OffOUT variable is assigned to the ReqOUT variable and returned to the service requestor, in case of an instance, o is assigned to r.



In our example, the ReqOUT variable of type Location is matched against the named instance room335 of type Room. As Room is a subtype of Location, the match succeeds because of the type check strategy **sub**, and the fixed value room335 is assigned to the variable. The service requestor will obtain this value after a successful service execution.

As the matching process is guided by the request's structure, it is possible that there are OffIN variables that are not reached by the matcher. However, for a correct service invocation, a concrete value has to be assigned to them. Thus, an arbitrary (preferably the default) value is assigned.

4.4 Needed Set Operations for the Complete Algorithm

The different cases from above show that dealing with variables requires three basic operations on object sets. Given two object sets s_1 and s_2, the following operations are needed:

- *disjunct.* Is the intersection $i = s_1 \cap s_2$ empty? I.e., are there any objects that are elements of both sets? We will abbreviate this operation as method `boolean disjunct(Set s1, Set s2)`.
- *subset.* Is s_1 a subset of s_2? I.e., is every object in s_1 also in s_2?. We will abbreviate this operation as method `boolean subset(Set s1, Set s2)`.
- *pickElement.* The task of theis operation is to return an arbitrary object o that is element of the intersection $s_1 \cap s_2$. We will abbreviate this operation as method `Instance pickElement(Set s1, Set s2)`.

All these operations are computable with the definition possibilities for sets presented in Section 3.1.

With these possibilities, we can construct the complete algorithm (see Listing 1.2). Its return value is `boolean+` as it can return `true`, `false`, `neutral`, and `missing`. `missing` is only used for partial results and cannot be the result of the whole match.

```
boolean+ match(r,o)
{
    //TypeCheckStrategy
    if (not type(r) <tcs(r)> type(o)) return false;

    //OffOUT
    if (o is OffOUT)
    {
        if (disjunct(o,r)) return false;
        if (subset(o,r)) goto :match;
        else return missing;
    }

    //OffIN
    if (o is OffIN)
    {
        if (disjunct(o,r))
        {
            o.assign(default(o));
            return false;
        }
        o.assign(pickElement(o,r));
        goto :match;
    }

    if (o not fulfills directConditions(r)) return false;

    matches = true;
    for each defined property condition p of r
    {
        if (o.p is defined) singleresult = match(r.p, o.p);

        if ((o.p is undefined) or (singleresult == missing))
        {
            //MissingStrategy
            if (ms(p) == assume_failed) singleresult = false;
            if (ms(p) == assume_fulfilled) singleresult = true;
            if (ms(p) == ignore) singleresult = neutral;
        }

        //ConnectingStrategy
        matches = matches <cs(p)> singleresult;
    }

    :match

    //ReqOUT
    if (r is ReqOUT)
    {
        if (matches == true)
        {
            if (o is OffIN) r.assign(assignment(o));
            if (o is OffOUT) r.connect(o);
            if (o is instance) r.assign(o);
        }
        else r.assign(null);
    }

    return matches;
}
```

Listing 1.2. Complete matching algorithm

5 Conclusion

In this paper, we have presented an approach for automatically matching semantic service descriptions. In contrast to existing approaches, it does not rely on an explicit signature matching step, but couples this task with the specification matching. This enables the algorithm to find *all* functionally suitable services even those, whose signatures do not match the request. Moreover, the messages needed for an automatic invocation are derived. To achieve this goal, the matcher compares state based service descriptions.To integrate the message flow into the state descriptions of pre- and postconditions, the concept of declarative object sets and the usage of variables in service descriptions have been introduced.

References

1. Klein, M., König-Ries, B., Obreiter, P.: Stepwise refinable service descriptions: Adapting DAML-S to staged service trading. In: Proc. of the First Intl. Conference on Service Oriented Computing, Trento, Italy (2003) 178–193
2. Paolucci, M., Kawmura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Proc. of the First International Semantic Web Conference, Sardinia, Italy (2002)
3. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: Proc. of Web Services: Modeling, Architecture and Infrastructure. Workshop in Conjunction with ICEIS2003, Angers, France (2003)
4. Sycara, K.P., Klusch, M., Widoff, S., Lu, J.: Dynamic service matchmaking among agents in open information environments. SIGMOD Record **28** (1999) 47–53
5. Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems **5** (2002) 173–203
6. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A semantic web approach to service description for matchmaking of services. In: Proc. of the Intl. Semantic Web Working Symposium (SWWS), Stanford, CA, USA (2001)
7. Gonzalez-Castillo, J., Trastour, D., Bartolini, C.: Description logics for matchmaking services. In: Proc. of the Workshop on Applications of Description Logics at KI-2001, Vienna, Austria (2001)
8. Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: A system for principled matchmaking in an electronic marketplace. In: Proc. of the Twelfth Intl. World Wide Web Conference, Budapest, Hungary (2003)
9. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proc. of the Twelfth Intl. World Wide Web Conference (WWW 2003), Budapest, Hungary (2003)
10. Li, L., Horrocks, I.: Matchmaking using an instance store: Some preliminary results. In: Proc. of the 2003 Intl. Workshop on Description Logics (DL'2003), poster paper, Rome, Italy (2003)
11. Klein, M., König-Ries, B.: Combining query and preference - an approach to fully automatize dynamic service binding. In: Short Paper at IEEE International Conference on Web Services, San Diego, CA, USA (2004)

Negotiation Among Web Services Using LOTOS/CADP

Gwen Salaün, Andrea Ferrara, and Antonella Chirichiello

DIS - Università di Roma ”La Sapienza”
Via Salaria 113, 00198 Roma, Italia
Contact salaun@dis.uniroma1.it

Abstract. It is now well-admitted that formal methods are helpful for many issues raised in the web service area. In a previous work, we advocated the use of process algebra to describe, compose and reason on web services at an abstract level. In this paper, we extend this initial proposal, which only dealt with behavioural aspects, to cope with the question of representing data aspects as well. In this context, we show how the expressive process algebra LOTOS (and its toolbox CADP) can be used to tackle this issue. We illustrate the usefulness of our proposal on an important application in e-business: negotiation among web services. The connection between abstract specifications and running web services is made concrete thanks to guidelines enabling one to map LOTOS and the executable language BPEL in both directions.

Keywords: Web Services, Formal Methods, LOTOS, CADP, Negotiation, BPEL.

1 Introduction

Web services (WSs) are distributed and independent pieces of code solving specific tasks which communicate with each other through the exchange of messages. Some issues which are part of on-going research in WSs are to specify them in an adequate, formally defined and expressive enough language, to compose them (automatically), to discover them through the web, to ensure their correctness, etc. Formal methods provide an adequate framework (many specification languages and reasoning tools) to address most of these issues (description, composition, correctness). Different proposals have emerged recently to abstractly describe WSs, most of which are grounded on transition system models [2,11, 18,10,14] and to verify WS description to ensure some properties on them [18,7, 17]. In a previous work [19], we advocated the use of process algebra (PA) [3] for WSs. Being simple, abstract and formally defined, PAs make it easier to specify the message exchange between WSs, and to reason on the specified systems (*e.g.* using bisimulation notions to ensure the correctness of composition).

In this initial proposal [19], we especially experimented the use of the simple process algebra CCS. However, CCS turns out to be only adequate for the specification of (and reasoning on) dynamic behaviours. What was missing in this proposal was to handle *data*. This allows a much finer (less abstract) level of specification, which is clearly needed in some cases. In this paper, we argue that the process algebra LOTOS [12] and its toolbox CADP are useful respectively to describe WSs and to reason on them. We also propose a two-level description of WSs: an abstract one (using LOTOS) and an executable one (using WSDL and BPEL) Following such an approach, we can develop WSs considering the formal and verified specification as a starting point. In the other direction, we can abstract a deployed system to a description in LOTOS. The interest of such an abstract description is that the formality of this language and its readily existing tools enable one to validate and verify specifications through animation and proofs of temporal properties.

To illustrate the interest of such an approach in WSs, we focus on the problem of *negotiation* in which both data and dynamic aspects have to be dealt with. The perspective of intelligent/automated WSs which would be able to automatically perform the necessary negotiation steps to satisfy their user's request in the most satisfactory possible way emerged from artificial intelligence and multi-agent systems. This problem is a typical example of services involving both data (prices, goods, stocks, etc) and behaviours. Negotiation issues appear when several participants (clients and providers) have to interact to reach an agreement that is beneficial to all of them. Our goal is to show how LOTOS/CADP may be used to ensure trustworthy and automated negotiation steps.

The organization of this paper is as follows. First, we introduce in Section 2 the different entities involved in negotiation. Section 3 presents the LOTOS language and its toolbox CADP. They are used in Section 4 to describe negotiating processes at an abstract level, and to reason on them. Section 5 gives some guidelines to map LOTOS specifications and BPEL code in both directions. Related works are introduced in Section 6 and compared with the current proposal. Finally, we draw up some concluding remarks in Section 7. This paper is a shorter version of [20] in which the reader can find much more details.

2 What Does Negotiation Involve?

In this section, we introduce what is involved in negotiation cases. Specification and implementation of such aspects are resp. described in Sections 4 and 5.

Variables. They represent entities on which processes should negotiate, *e.g.* a price. Many variables may be involved in a negotiation at the same time (availability of different products, fees, maximum number of days for a delivery).

Constraints. They represent conditions to respect (called invariants as well) while trying to reach an agreement. Such an invariant is actually a predicate which can be evaluated replacing free variables with actual values. For a requester who is trying to buy by auction a product, such an invariant could be that (s)he is ready to pay €300 at most with a delivery within 10 days, or to accept a possible late delivery if there is a price reduction of 10% at least.

Exchanged information. To reach an agreement, both participants should send values to the other. A simple case is a price, but they can also exchange more advanced constructs (a record of values, a constraint on a value, etc).

Strategies. *"An agent's negotiation strategy is the specification of the sequence of actions (usually offers or responses) the agent plans to make during the negotiation."* [16]. Strategies may take into account other considerations. For instance, a participant can try to reach an agreement as soon as possible, or to minimize a price. Therefore, strategies are related to minimizing or maximizing objective functions.

In this proposal, we discard lots of possible variants which possibly appear in negotiations such as evolution of the constraints (a requester who is not finding a product less than €300, modifies his/her constraint to pay up to €320 from a certain point in time) or the level of automation (complete automation or intervention of human people). Consequently, all these variations may be combined and may end up to many possible scenarios of negotiation. It is obvious that no negotiation process is better than another. The right process should be selected depending on the bargaining context.

3 LOTOS and CADP in a Nutshell

LOTOS is an ISO specification language [12] which combines two specification models: one for static aspects (data and operations) which relies on the algebraic specification language ACT ONE and one for dynamic aspects (processes) which draws its inspiration from the CCS and CSP process algebras.

Abstract Datatypes. LOTOS allows the representation of data using algebraic abstract types. In ACT ONE, each *sort* (or datatype) defines a set of *operations* with arity and typing (the whole is called *signature*). A subset of these operations, the *constructors*, are sufficient to create all the elements of the sort. *Terms* are obtained from all the correct operation compositions. *Axioms* are first order logic formulas built on terms with variables; they define the meaning of each operation appearing in the signature.

Basic LOTOS. This PA authorizes the description of dynamic behaviours evolving in parallel and synchronizing using rendez-vous (all the processes involved in the synchronization should be ready to evolve simultaneously along the same gate). A process P denotes a succession of actions which are basic entities representing dynamic evolutions of processes. An action in LOTOS is called a *gate* (also called event, channel or name in other formalisms). The symbol **stop** denotes an inactive behaviour (it could be viewed as the end of a behaviour) and the **exit** one depicts a normal termination. The specific i gate corresponds to an internal evolution.

Now, we present the different LOTOS operators. The prefixing operator G**;**B proposes a rendez-vous on the gate G, or an independent firing of this gate, and then the behaviour B is run. The nondeterministic choice between two behaviours is represented using []. LOTOS has at its disposal three *parallel composition* operators. The general case is given by the expression B_1 $|[G_1, ..., G_n]|$ B_2 expressing the parallel execution between behaviours $B1$ and $B2$. It means that $B1$ and $B2$ evolve independently except on the gates G_1, ..., G_n on which they evolve at the same time firing the same gate (they also synchronize on the termination **exit**). Two other operators are particular cases of the former one to write out interleaving $B_1|||B_2$ which means an independent evolution of composed processes B_1 and B_2 (empty list of gates), and full synchronization $B_1||B_2$ where composed processes synchronize on all actions (list containing all the gates used in each process). Moreover, the communication model proposes a multi-way synchronization: n processes may participate to the rendez-vous.

The operator **hide** G_1, ..., G_n **in** B aims at hiding some internal actions for the environment within a behaviour B. Consequently, the hidden gates cannot be used for the synchronization between B and its environment. The *sequential composition* $B_1 \gg B_2$ denotes the behaviour which executes B_2 when B_1 has successfully terminated (**stop** or **exit**). The *interruption* B_1 [> B_2 expresses that the B_1 behaviour can be interrupted at any moment by the behaviour B_2. If B_1 terminates correctly, B_2 is never executed.

Full LOTOS. In this part, we describe the extension of basic LOTOS to manage data expressions, especially to allow value passing synchronizations. A process is parameterized by a (optional) list of formal gates $G_{i \in 1..m}$ and a (optional) list of formal parameters $X_{j \in 1..n}$ of sort $S_{j \in 1..n}$. The full syntax of a process is the following:

$$\textbf{process}\ P\ [G_0, ..., G_m]\ (X_0{:}S_0, ..., X_n{:}S_n) : func := B\ \textbf{endproc}$$

where B is the behaviour of the process P and $func$ corresponds to the functionality of the process: either the process loops endlessly (**noexit**), or it terminates (**exit**) possibly returning results of sort $S_{j \in 1..n}$ (**exit**(S_0, ..., S_n)).

Gate identifiers are possibly enhanced with a set of parameters (offers). An *offer* has either the form $G!V$ and corresponds to the emission of a value V, or the form $G?X{:}S$ which means the reception of a value of sort S in a variable X. A single rendez-vous can contain several offers. A behaviour may depend on Boolean conditions. Thereby, it is possible that it be preceded by a guard [*Boolean expression*] $\rightarrow B$. The behaviour B is executed only if the condition is true. Similarly, the guard can follow a gate accompanied with a set of offers. In this case, it expresses that the synchronization is effective only if the Boolean expression is true (*e.g.*, $G?X$`:Nat[`X`>3]`). In the sequential composition, the left-hand side process can transmit some values (**exit**) to a process B (**accept**):

$$...\ \textbf{exit}(X_0, ..., X_n) \gg \textbf{accept}\ Y_0{:}S_0, ..., Y_n{:}S_n\ \textbf{in}\ B$$

To end this section, let us say a word about CADP[1] which is a toolbox for protocol engineering. It particularly supports developments based on LOTOS specifications. It proposes a wide panel of functionalities from interactive simulation to formal verification techniques (minimization, bisimulation, proofs of temporal properties, compositional verification, etc).

[1] `http://www.inrialpes.fr/vasy/cadp/`

4 Negotiation Using LOTOS/CADP

In this section, we do not argue for a general approach specifying any possible case of negotiation (even though many negotiation variants can be described in LOTOS). Our goal is to illustrate the use and interest of LOTOS/CADP for negotiating services. Consequently, we introduce our approach on a classical case of peer-to-peer sale/purchase negotiation involving one client (it works with more clients but we explain with one) and many providers. The goal of a formal representation and hence of automated reasoning is to prove properties so as to ensure a correct and safe automated negotiation between involved processes.

An assumption in this work is that we have a privileged view of all the participants and their possible behaviours (particularly in case of reverse engineering approach). Consequently, from our point of view, processes are glass boxes. This hypothesis is essential in any situation where we want to reason on interacting processes. An alternative approach would be to consider processes as black boxes and to reason on visible traces. However, it is almost impossible given such inputs, in which little information is available, to ensure critical properties.

Aspects involved in negotiation and itemized in Section 2 may be encoded in LOTOS in different ways. Let us show now an outline of the description we propose (see further for detailed explanations).

• Datatypes are defined using ACT ONE algebraic specifications and are afterwards used to type **variables** locally defined as parameters of processes.

• A specific datatype (`Inv`) is defined to describe **constraints** to be respected by participants while negotiating.

• **Exchanged information** are represented by variables. They can represent simple values (natural numbers) or more complex ones (a constraint on multi-type values). Values are exchanged between processes along gates.

• **Strategies** are much trickier because they are encoded either in the dynamic processes – the way a participant proposes some values and more generally interacts with other participants – and in the local variables managed by processes – initial value, computation of the next ones, constraints to be respected.

In this section, we introduce the main ideas of the negotiation situation at hand and illustrate them with short pieces of specification borrowed from the comprehensive one[2]. Additionally, this section does not make any assumption regarding the development approach (design approach or reverse engineering).

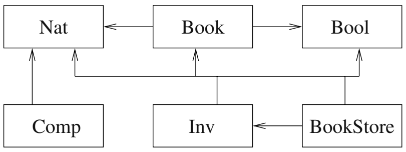
Case Study. We illustrate our proposal using an on-line auction sale. This case involves a client who interacts with book providers (one after the other) to purchase a book respecting some constraints. In the following, we choose to simplify the case for the sake of readability and comprehension (purchase of a single book, negotiating only the price). However, our approach works for more complex cases and many negotiation variants, as it has been experimented[3]: generalization of data management, handling of bigger sets of books, negotiations on several values and accordingly encoding of more complex invariants.

[2] `http://www.dis.uniroma1.it/~salaun/LOTOS4WS/BASIC`

[3] `http://www.dis.uniroma1.it/~salaun/LOTOS4WS/GEN`

Data Descriptions. First, the ADT abilities of LOTOS are useful to represent the data variables handled during the negotiation as well as the more complex data managed by processes. With regards to our negotiation problem, we need the datatypes gathered on the left-hand side in Figure 1 with their import links (Nat and Bool are already defined in the CADP library). A book is represented using four values: an identifier, a price, the delivery time, the possibility to return the book or not. A bookstore is described as a set of books. Each entry in the bookstore also contains the number of available books and the invariant to be respected, then deciding of the offer acceptance or refusal. Several operations are defined to access, modify and update these datatypes.

Constraints are represented using a generic datatype. They are encoded using the `Inv` sort which defines as many invariants as needed and a `conform` function evaluating invariants with actual values. In the example below (right-hand side in Fig. 1), the invariant means that the maximum price to be paid is €3 (client condition), the corresponding meaning of the `conform` function is written out using the judicious comparison operator on natural numbers. Another example shows how more complex invariant can be expressed (the price has to be less than or equal to three, the delivery within 5 days and the return possible).



```
pMax3 (*! constructor *) : -> Inv
conform : Nat, Inv -> Bool
...
eqns  forall p: Nat, b: Book
ofsort Bool     (* type of the result *)
  conform(p, pMax3) = (p<=3);
  conform(b, myInv) = (price(b)<=3) and
           (days(b)<5) and returnok(b);
```

Fig. 1. Datatype dependence graph (left) and sample of the datatype for constraint descriptions (right)

The computation of the initial value to be negotiated and of the next value to be proposed (*e.g.* adding up one to the price) are also encoded in a generic way using the `Comp` sort. For example, it is done for the price using some operations extending the `Nat` existing datatype.

Negotiating Processes. Now, let us define the processes involved in the negotiation steps. In the current negotiation example, we advocate a fully delegated negotiation service (the user does not intervene into search of an agreement and negotiation rounds). In Figure 2, boxes correspond to nested processes and lines to interactions between processes (they hold the gates used by the processes to communicate). Nested processes mean that, at one moment, there is only one control flow: one parent process instantiates another one and waits for the end of its fork before continuing its behaviour. It is compulsory in LOTOS because the use of an intermediate process is the single way to express a looping behaviour inside a bigger one.

The `Controller` process is run by the client with judicious parameters: the identifier of the book to purchase, an initial value to be proposed for the book

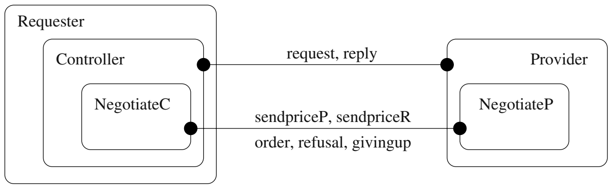


Fig. 2. Processes involved in our negotiation process

while negotiating, an invariant to be respected during the negotiation steps, a function computing the next proposal. Its *exit* statement is accompanied with a Boolean which indicates an order or a failure of the negotiation rounds. On the other hand, each provider has at its disposal a set of books, a function to compute a first value (to be proposed while negotiating) from the actual price as stored in the book set, and a function to compute the price of the book for the next negotiation round. Many parts of our encoding favour reusability, but values involved in the negotiation remain close to the current problem (book identifier, book price, bookstore). Accordingly, adjustments of these parameters are needed to reuse such negotiating processes in another context.

Let us sketch the behaviour of the controller. First, it initiates the interaction with a provider by sending the identifier of the book to be sought, then receives a Boolean answer denoting the availability of the book. A false reply implies a recursive call. A true response induces the instantiation of the negotiation process `NegotiateC` and the waiting for an answer accompanied with a status which is true in case of an order (and then exit alerting the client of the success) and false in case of a failure (recursive call to try another provider). At any moment, the controller may give up and exit alerting the client that every negotiation attempt has failed. Each provider has to be willing to interact with such a controller, and then is made up of the symmetrical behaviour.

```
process Controller
   [request,reply,order,refusal,sendpriceP,sendpriceR,givingup]
   (ref: Nat, pi: Nat, inv: Inv, computfct: Comp): exit(Bool) :=

   request!ref; reply?b:Bool;
   (
     ([b] ->
      NegotiateC[order,refusal,sendpriceP,sendpriceR,givingup]
      (pi, inv, computfct) >> accept status: Bool in
        (
          [not(status)] ->          (* case of a failure *)
            Controller[request,reply,order,refusal,sendpriceP,...]
              (ref, pi, inv, computfct)
          []
          [status] -> exit(true)     (* case of an order *)
        ))
     []
     ([not(b)] ->         (* let us try another bookstore *)
       Controller[request,...]
```

```
        (ref, pi, inv, computfct))
   )
   []
   exit(false)   (* the controller stops the negotiation *)

endproc
```

Below, we show the body of the `NegotiateC` process in charge of the negotiation on behalf of the controller. Negotiation steps are composed of classical information exchanges. One of both participants proposes a price to the other. The other participant accepts whether this price satisfies its local constraints. Otherwise, it refuses and calls recursively itself. The caller updates its local value (price here) for the next proposal using the local function dedicated to such computations. At any moment, the negotiation may be abandoned by one of the participants.

```
process NegotiateC [order,refusal,sendpriceP,sendpriceR,givingup]
   (curp: Nat, inv: Inv, computfct: Comp): exit(Bool) :=

   sendpriceP?p:Nat;         (* the provider proposes a value *)
   ( [conform(p,inv)] -> order;exit(true)       (* agreement *)
     []
     [not(conform(p,inv))] -> refusal;
       NegotiateC[order,refusal,sendpriceP,sendpriceR,givingup]
        (curp, inv, computfct) )
   []                  (* proposal of a value to the provider *)
   ( [conform(curp,inv)] -> sendpriceR!curp;
     (
       order; exit(true)        (* agreement *)
       []
       refusal; NegotiateC[order,refusal,sendpriceP,sendpriceR,givingup]
                   (compute(curp,computfct),inv,computfct)
     ) )
   []    (* the client stops because results not satisfactory *)
   givingup; exit(false)

endproc
```

Let us introduce an example of a concrete system in which one requester is seeking a book among three possible providers. First of all, the needed data are described using appropriate algebraic terms. A simple example of a bookstore is the following one containing eight books (we experimented our approach with stores handling tens of books). For illustration purposes, the first book (identified by 0) has three copies still available and has to be sold at least €3.

```
  bs1: BookStore =
      add2BS(book(0,2,6,true), 3, pMin3,
        ...
          add2BS(book(7,3,2,false), 3, pMin5, emptyBS)...)
```

A lightweight view of the system is now given. This instance (and some variants of it modifying the number of possible providers) is used in the next subsection to assess the reasoning capabilities. The requester is seeking the book identified by `0`, with `2` as initial negotiating value, `pMax3` as price constraint and a computation of the next value obtained adding up one.

```
Requester [request,reply,order,refusal,...] (0,2,pMax3,add1)
  |[request,reply,order,refusal,sendpriceP,sendpriceR,givingup]|
(
   Provider [request,reply,order,refusal,...] (bs1,times2,minus1)
    ||| Provider[...] (bs2,times2,minus1) |||  Provider[...] (...)
)
```

Reasoning. Our purpose here is to validate our negotiation system and to verify some properties on it. All the steps we are going to introduce have been very helpful to ensure a correct processing of the negotiation rounds. Several mistakes in the specification have been found out (in the design, in the interaction flows, in the data description and management). Simulation has been carried out and proofs have been verified using EVALUATOR (a CADP on-the-fly model checker) and the prototype version CAESAR 6.3 [9] building smaller graphs than in the current distribution of CADP. Safety, liveness and fairness properties written in μ-calculus have been verified (examples are resp. introduced below).

A first classical property is the absence of deadlocks (actually there is one deadlock corresponding to the correct termination leading in a sink state). Another one (P1) ensures than an agreement is possible (because it exists one branch labelled by `ORDER` in the global graph). The property P2 expresses that the firing of every `SENDPRICER` gate (also checked with `SENDPRICEP`) is either followed by an order or a refusal. P3 verifies that after the firing of a `REPLY!TRUE` action, either `SENDPRICEP` and `SENDPRICER` are fireable.

```
< true* . "ORDER" > true                                      (P1)
[true* . "SENDPRICER!*"] <("ORDER" or "REFUSAL")> true        (P2)
[true* . "REPLY!TRUE"] <"SENDPRICEP" and "SENDPRICER"> true   (P3)
```

We end with a glimpse (Tab. 1) of experimentations we carried out on this system to illustrate from a practical point of view the time and space limits of our approach (and of the use of CADP). We especially dealt with one user since the goal is to ensure properties for a given user even if proofs can be achieved with several ones as showed below (however in this case the existence of an agreement P1 is meaningless). We used a Sun Blade 100 equipped with a processor Ultra Sparc II and 1.5 GB of RAM. To summarize, though we can reason on rather realistic applications, our approach is limited by the state explosion problem due especially in our case to the increase of: the number of processes (and underlying new negotiation possibilities), the number of values to be negotiated (if constraints are relaxed, negotiations can be hold on more values), the size of data (above all the number of books managed by each provider).

Table 1. Practical assessment of the approach

	Number of processes	Nb of states	Nb of trans.	P1	P2	P3
Example 1	(1 user & 1 provider)	32	47	3.84s	2.15s	2.21s
Example 2	(1 user & 5 providers)	2,485	5,124	4.88s	3.57s	3.57s
Example 3	(1 user & 7 providers)	17,511	42,848	4.64s	27.70s	27.35s
Example 4	(1 user & 8 providers)	35,479	88,438	5.02s	101.96s	102.27s
Example 5	(1 user & 10 providers)	145,447	374,882	5.10s	1326.94s	1313.16s
Example 6	(2 users & 4 providers)	300,764	944,394	5.31s	117.41s	117.79s

5 Two-Way Mapping Between LOTOS and WSDL/BPEL

First of all, we highlight once again the need of equivalences between behaviours written out in both languages, consequently each one can be obtained from the other. In this section, our goal is not to give a comprehensive mapping between both languages, that is a mapping taking into account all LOTOS and BPEL concepts. Herein, we focus particularly on the notions that we need for the negotiation problem as mentioned in the LOTOS specification introduced in the previous section. For lack of space, it is not possible to introduce BPEL. Accordingly, the reader who is not used with the language should refer to [1].

Two related works are [7,8]. Comparatively, our attempt is more general at least for three reasons: (i) the expressiveness of properties is better in CADP thanks to the use of the μ-calculus (*e.g.* only LTL in [8]), (ii) our abstract language does not deal only with dynamic behaviours; we can specify advanced data descriptions and operations on them at the abstract level (more complex than in [8] where operations cannot be modelled as an example) as well as at the executable one, (iii) we prone a mapping in two ways, useful to develop WSs and also to reason on deployed ones (the latter direction was the single goal of mentioned related works).

LOTOS Gates/Rendez-vous and BPEL Basic Activities. The basic brick of behaviour in LOTOS, the so-called gate, is equivalent to messages described in WSDL which are completely characterized using the *message*, *portType*, *operation* and *partnerLinkType* attributes. Most of the time abstract gates are accompanied of directions (emissions or receptions) because they are involved in interactions. Each action involved in a synchronization maps the four previous elements, and conversely. It is impossible to specify independent evolution of one process in BPEL: everything is interaction. Therefore, synchronizing gates are equivalent to BPEL interactions, especially in our case in which we deal only with binary communication (the core of the BPEL process model being the notion of peer-to-peer interaction between partners).

An abstract action accompanied with a reception (noted with a question mark '?' in LOTOS) may be expressed as a reception of a message using the *receive* activity in BPEL. On the other side, an emission (noted with an exclamation mark '!') is written in BPEL with the *asynchronous invoke* activity. At the abstract level, an emission followed immediately by a reception may be encoded using the BPEL *synchronous invoke*, performing two interactions (sending a request and receiving a response). On the opposite side, the complementary reception/emission is written out using a *receive* activity followed by a *reply* one.

Data Descriptions. There are three levels of data description: type and operation declarations, local variable declarations, data management in dynamic behaviours. We discuss the two first ones in this part and we postpone the latest one in the next subsection.

First, LOTOS datatypes, and operations on them, are specified using algebraic specifications. In BPEL, types are described using XML *schema* in the WSDL files. Elements in the schema can be simple (lots are already defined) or complex types. Data manipulation and computation (equivalent to operations in LOTOS) is defined in BPEL using the *assign* activity, and particularly using adequately XPath (and XPath Query) to extract information from elements. XPath expressions can be used to indicate a value to be stored in a variable. Within XPath expressions, it is possible to call several kinds of functions: core XPath functions, BPEL XPath functions or custom ones. Complex data manipulation may be written using XSLT, XQuery or JAVA (*e.g.* BPELJ makes it possible to insert Java code into BPEL code) to avoid a tricky writing with the previous mentioned means. Another way to describe and manage data is to use a database and corresponding statements to accessing and manipulating stored information. The data correspondence between both languages is not straightforward, although expressiveness of both data description techniques makes it possible to map all specifications from one level to the other.

In LOTOS, variables are either parameters of processes or parameters of a gate. In BPEL, variables can represent both data and messages. They are defined using the *variable* tag (global when defined before the activity part) and their scope may be restricted (local declarations) using a *scope* tag. A (WSDL) *message* tag corresponds to a set of gate parameters in LOTOS. A *part* tag matches with a parameter of a gate in LOTOS. In LOTOS, only process parameters need to be declared (not necessary for gate variables) whereas in BPEL either global and local variables involved in interactions have to be declared.

LOTOS Dynamic Constructs and BPEL Structured Activities. First of all, the *sequence* activity in BPEL matches with the LOTOS prefixing construct ';'. Intuitively, the LOTOS choice (possibly multiple) corresponds either to the *switch* activity defining an ordered list of *case* (a *case* corresponds to a possible activity which may be executed) or to the *pick* one. Let us clarify the two possible equivalences between LOTOS and BPEL depending on the presence or not of guards: (i) *absence of guards*, a choice is translated with a *pick* activity with *onMessage* tags, (ii) *presence of guards*, the mapping is straightforward using the *switch* operator. The termination used in the LOTOS specification maps with the end of the main sequence in BPEL.

Recursive process calls match with the *while* activity. The condition of the *while* is the *exit* condition of the recursive process. Sometimes, abstract recursive behaviours match BPEL non recursive services. It is the case when dealing with the notion of *transaction* (defined as a complete execution of a group of interacting WSs working out a precise task) because in this context, only one execution of a process is enough (each transaction corresponds to a new invocation then instantiation of each involved service).

Now, let us focus on the constructs involving dynamic behaviours and data: parameters of messages and processes (in case of recursive calls for the latter), conditions of guards, local definition and modification of data. We have already discussed in the subsection dedicated to the data descriptions how to describe parameters of gates and processes in BPEL. LOTOS guards correspond to BPEL *case* tags of the *switch* construct. The BPEL *assign* tag, and more precisely the *copy* tag matches the four next cases in LOTOS: (i) *let* X_i:T_i=V_i *in* B means the initialization of variables X_i of types T_i with values V_i ($\forall i \in 1..n$) in the behaviour B, (ii) B_1*; exit(*Y_i*)* $\gg$ *accept* X_i:T_i *in* B_2 denotes the modification of variables X_i (replaced by new values Y_i), (iii) *P(*X_i*)* is an instantiation of a process or a recursive call meaning assignments of values X_i to the parameters of the process P, (iv) *send*(X_i) corresponds to an emission of data expressions which have to be built and assigned to variable X_i before sending.

LOTOS and BPEL Processes. BPEL services and LOTOS processes correspond to each other. An external view of such interacting WSs shows processes/services running concurrently. Such a kind of global system in LOTOS is described using a LOTOS main behaviour made up of instantiated processes composed in parallel and synchronizing together. Let us observe that the main LOTOS specification does not match with a BPEL process. The correspondence is that each LOTOS instantiated process, therefore pertaining to the global system mentioned previously, matches a BPEL WS. Accordingly, the architecture of the specification is preserved.

In BPEL, variables, messages/operations/port types/partner links and its main activity match respectively in LOTOS process parameters, action declarations and its behaviour. Due to the possible correspondence between *while* loops and recursive processes, an adequate use of the *scope* activity is needed to map the local variables of such nested processes.

The Negotiation Case in BPEL. The negotiation issue focused on in this paper has been implemented in BPEL and the resulting services work correctly. In this experimentation, we used guidelines in the development stage way, but the opposite direction could be tackled as well using them. Experimentations have been carried out using Oracle BPEL Process Manager 2.0[4] that enables one to design, deploy and manage BPEL processes, and BPEL Designer (an Eclipse plug-in) which is a graphical tool to build BPEL Processes. For lack of space, it is impossible to introduce pieces of BPEL code in the paper. However, all the WSDL and BPEL files implemented for this problem are available on-line[5].

6 Related Work

We split this survey of related works into two separate parts: (i) use of formal methods for WSs, (ii) contribution *wrt* the negotiation issue. We first outline

[4] `http://www.collaxa.com/`

[5] `http://www.dis.uniroma1.it/~salaun/LOTOS4WS/BPEL`

some existing proposals arguing for the use of formal methods as an abstract way to deal with WSs (especially description, orchestration, reasoning). At this abstract level, lots of proposals originally tended to describe WSs using semi-formal notations, especially workflows [15]. More recently some more formal proposals grounded for most of them on transition system models (LTSs, Mealy automata, Petri nets) have been suggested [11,18,10,2,14]. Regarding the reasoning issue, works have been dedicated to verifying WS description to ensure some properties of systems [8,18,7,17]. Summarizing these works, they use model checking to verify some properties of cooperating WSs described using XML-based languages (DAML-S, WSFL, BPEL, WSCI). Accordingly, they abstract their representation and ensure some properties using ad-hoc or existing tools.

In comparison to these existing works, the strenght of our alternative approach is to work out all these issues (description, composition, reasoning) at an abstract level, based on the use of expressive (especially compared to the former proposals) description techniques and adequate tools (respectively LOTOS and CADP), while keeping a two-way connection with the executable layer (BPEL).

A lot of research works about negotiation have been proposed in different research domains and aim at working out different issues. Main issues in negotiation are to describe negotiating systems, to verify the existence of an agreement (and possibly other properties) and to speed up its reaching. Most of the proposals belong to the multi-agent system area. Let us illustrate with former works dedicated to describe and automate negotiations [23,22,6]. As an example, in [22], the authors proposed to use logic-based languages for multi-agent negotiation. Then, they can verify from a given history that an agreement has been reached and for a given protocol that an agreement will be reached.

Other existing works advocate some description frameworks, see [4,21] as examples, to represent all the concepts (profiles, policies, strategies, etc) used in negotiations and to ensure trust negotiations. In [21], the authors advocated a model-driven trust negotiation framework for WSs. Their model is based on state machines extended with security abstractions. Their approach allows dynamic policy evolution as well. Another proposal related to WSs [5] suggested a flexible meta-model based on contracts for e-negotiation services. From this approach, negotiation plans can be derived and implementations can be performed in recent WS platform like .NET. [13] proposed some experiments on automated negotiation systems using WSs. The authors implemented a RFQ-based simple negotiation protocol using the BPEL executable language.

Compared to these existing works, we first propose a formal representation of negotiation situations. The formality of our approach implies the ability to reason on it to ensure correctness of interacting processes (either developing concrete WSs or abstracting away from an executable one) particularly to prove the existence of an agreement. We also claim that the use of WSs is adequate to negotiating systems due to many of their characteristics: interoperability, autonomy and automation, expressiveness of description, deployment on the web, etc.

7 Concluding Remarks

The recent advent of WSs raised many promising issues in the web computing area. In this paper, we emphasized that the use of an expressive, formal, tool-equipped PA is convenient to abstractly describe and reason on negotiating WSs. It was reinforced in this paper through the double-mapping we introduced to map abstract and executable layers. On a wider scale, our two-level (abstract and concrete) approach is worthy because it can be used with any kind of interacting WSs and not only for the negotiating ones on which we focused on herein.

When using PA for WSs, during the choice of the description language, an adequate trade-off should be chosen between expressiveness of the calculus and verification abilities of its support tool. For example, LOTOS is adequate to represent negotiation aspects, but is therefore limited at the verification level due to the state explosion problem ensuing the management of data expressions. All the same, we stress that (i) we already tackled and verified realistic applications, (ii) CADP is one of the most efficient tool dealing with automated reasoning on input formats involving mixed descriptions (behaviours and complex data).

Our approach can easily be generalized in many ways. First, during the design stage from PA to WSs, some negotiation patterns can be extracted, simplifying the reusability of our process description. Hence, adjustments should be performed to reuse these processes for other negotiation variants. As an example, we have already experimented multiple issue negotiation (price, time delivery, possible return, etc) implying exchanges of several values and invariants involving several parameters. Secondly, for reverse engineering purposes, guidelines to abstract BPEL code are enough to tackle most of the negotiation situations. Though, they may be complemented to deal with more complex variants.

A first perspective is to propose a methodology formally defining how to use LOTOS/CADP in the context of WSs. A possible application would be the development of certified WSs from scratch to executable BPEL code, with intermediate steps of specification and verification using LOTOS/CADP. A related perspective is to develop a tool automating the two-way mapping to obtain LOTOS or BPEL skeletons. This implementation should be performed from the guidelines (only experimented manually so far) defined in this paper.

Acknowledgments. The authors thank Lucas Bordeaux and Marco Schaerf for judicious comments and fruitful discussions we had on this work, as well as Nicolas Descoubes, Hubert Garavel and Wendelin Serwe for their help on doing experimentations with CADP. This work is partially supported by Project ASTRO funded by the Italian Ministry for Research under the FIRB framework (funds for basic research).

References

1. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Specification: Business Process Execution Language for Web Services Version 1.1. 2003.

2. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic Composition of E-services That Export Their Behavior. In *Proc. of ICSOC'03*.
3. J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra.* Elsevier, 2001.
4. E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-chi: An XML Framework for Trust Negotiations. In *Proc. of CMS'03*.
5. D. K. W. Chiu, S.-C. Cheung, and P. C. K. Hung. Developing e-Negotiation Process Support by Web Service. In *Proc. of ICWS'03*.
6. S. S. Fatima, M. Wooldridge, and N. R. Jennings. An Agenda-based Framework for Multi-issue Negotiation. *Artificial Intelligence*, 152(1):1–45, 2004.
7. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based Verification of Web Service Compositions. In *Proc. of ASE'03*, pages 152–163, Canada, 2003. IEEE Computer Society Press.
8. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. of WWW'04*.
9. H. Garavel and W. Serwe. State Space Reduction for Process Algebra Specifications. In *Proc. of AMAST'04*.
10. R. Hamadi and B. Benatallah. A Petri Net-based Model for Web Service Composition. In *Proc. of ADC'03*.
11. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: a Look Behind the Curtain. In *Proc. of PODS'03*.
12. ISO. LOTOS: a Formal Description Technique based on the Temporal Ordering of Observational Behaviour. Technical Report 8807, International Standards Organisation, 1989.
13. J. B. Kim, A. Segev, and M. G. Cho A. K. Patankar. Web Services and BPEL4WS for Dynamic eBusiness Negotiation Processes. In *Proc. of ICWS'03*.
14. A. Lazovik, M. Aiello, and M. P. Papazoglou. Planning and Monitoring the Execution of Web Service Requests. In *Proc. of ICSOC'03*.
15. F. Leymann. Managing Business Processes via Workflow Technology. Tutorial at VLDB'01, Italy, 2001.
16. A. R. Lomuscio, M. Wooldridge, and N. R. Jennings. A Classification Scheme for Negotiation in Electronic Commerce. *International Journal of Group Decision and Negotiation*, 12(1):31–56, 2003.
17. S. Nakajima. Model-checking Verification for Reliable Web Service. In *Proc. of OOWS'02, satellite event of OOPSLA'02*.
18. S. Narayanan and S. McIlraith. Analysis and Simulation of Web Services. *Computer Networks*, 42(5):675–693, 2003.
19. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. In *Proc. of ICWS'04*.
20. G. Salaün, A. Ferrara, and A. Chirichiello. Negotiation among Web Services using LOTOS/CADP. Technical Report 13.04, DIS - Università di Roma "La Sapienza", 2004. Available on the G. Salaün's webpage.
21. H. Skogsrud, B. Benetallah, and F. Casati. Trust-Serv: Model-Driven Lifecycle Management of Trust Negotiation Policies for Web Services. In *Proc. of WWW'04*.
22. M. Wooldridge and S. Parsons. Languages for Negotiation. In *Proc. of ECAI'00*.
23. G. Zlotkin and J. S. Rosenschein. Mechanisms for Automated Negotiation in State Oriented Domains. *Journal of Artificial Intelligence Research*, 5:163–238, 1996.

An Investigation on Using Web Services for Micro-Payment

William Song[1], Deren Chen[2], and Jen-Yao Chung[3]

[1] Computer Science, Durham University, Durham, UK, w.w.song@dur.ac.uk
[2] Computer Science, Zhejiang University, Zhejiang, CHINA, drchen@zju.edu.cn
[3] IBM T.J. Watson Research Center, New York, USA, jychung@us.ibm.com

Abstract. E-commerce development and applications have been bringing the Internet to business and marketing and reforming our current business styles and processes. As Business to Business (B2B) has been a main stream of the e-commerce activities for a decade, business to customer (B2C) is still a new area to explore. Among the business to customer activities, a strong impact is security, i.e., how to protect secrets of various parties in business transactions. Different security methods apply to different businesses and meet different customers' needs. The methods with low cost and easy to implement may be less secure, whereas the methods providing high level security can be unaffordable to many small enterprises and individuals. In particular, people, while browsing or surfing on the Web, wish to pay a very small amount and "buy" (perhaps just download) an interesting item from the Web. The "click-and-pay" or "click-and-buy" does not require high security protection and should be convenient to use (buy and pay). Micro-payment issue emerges from the context to meet the need of applicability of low secure and cost in e-purchase. Payments of small amounts may enable many exciting applications on the Internet, such as selling information, small software and services. Users are willing to pay a small amount of money for the information, software and services. While the payment is small, expensive security protection is unnecessary. What users require may include negligible delay, rapid click-through, and automatic billing. Micro-payment is a service intensive activity, where many payment tasks involve different forms of services, such as payment method selection for buyers, security support software, product price comparison, etc. In this paper, we intend to investigate and analyze various aspects of micro-payment system using the web service technology. We will discuss business patterns for micro-payment systems using Web Services approaches.

1 Introduction

Use of the Internet has reshaped people's life in many ways. At the beginning, people use the Web as a huge information repository to search for what they need. Nowadays, people publish everything to the Web. On the Web, they advertise, they publish articles, they play games, they go shopping, and they even gamble. Two decades ago, people started doing business through the Internet. EDI (electronic data exchange) has been a standard for transferring data between two large enterprises in business transactions.

Nowadays, the Internet and the Web are reshaping people's business life. These terms, like e-commerce, e-procurement, e-payment, e-market, have illustrated a large number of different requirements, which need to be met in various e-business technology development.

1.1 New Market Requirements

Furthermore, e-business technology development brings the people's activities to a new stage. They can pay for their consumption. However, due to the high transaction expenses required by banks for using e.g. credit cards to buy things and unnecessarily high security cost, researchers and practitioners are looking for a payment method, which process purchases in small amounts, for example, less than 1 USD.

Undoubtedly, these requirements exist substantially in market. For instance, now in Japan, youngsters do not want to buy a pack of music CD. Instead, they want to pay to download their favorite songs from the Web. They buy the songs by downloading them from the Web. Even more they just buy to listen to a song from the Web. Similarly, you may like to pay for a piece of timely, important news. You may like to pay for downloading a scientific paper copy from a journal in a library. You even like to pay for accomplishment of regular formality, for example filling an application form for driving license online rather spending your time going to related department. Clearly, these payments are small, ranging from half to 5 USD. We call such payment micro-payment, or mini-payment.

1.2 Technology Needs

In micro-payment systems, simple authentication of the user is required. There have been several proprietary systems proposed for micro-payment on the Internet. Most of these involve setting up an account with a micro-payment service. The subscriber then prepays an amount into an account or registers it with a credit card account. The user can then go to service or goods providers that are subscribers to the system, and make purchases. When sufficient amounts have been exchanged, the merchant can make a settlement request to the micro-payment service provider.

The authentication part or (sometime called) certificate part is mainly required by the merchant side when a buyer is going to pay his or her electronic "money" for some items from the merchant site (usually on an e-market). This authentication is usually provided by the Internet Service Providers (ISP), banks, or authorized financial organizations. The security issue is critical here in the authentication process.

E-marketplace is also an important design in micro-payment application development. The e-marketplace, first, is the web site where buyers visit, choose and purchase items. Second, it is the place, where merchants upload, illustrate, and sell their goods. An e-marketplace can be owned by and used for just one large Internet Content Provider (ICP), such as amazon.com, but more often than not, it is a common place maintained for many ICPs to exhibit and sell their goods and services.

In addition, the billing systems need to be managed at the buyer's, the merchant's, and most importantly at the ISP's. The transaction security from one side to another needs to be carefully handled. We will discuss these issues in detail later.

1.3 Web Service Issues

Web Service is a piece of software that makes itself available over the Web and uses a standardized XML messaging system [3]. Its significance lies in composition of software for various purposes through the Web for users who attempt to accomplish certain tasks with the composed services. A micro-payment system requires various software components, such as security supporting programs, payment methods that buyers can select, e-wallet, etc. to work together to accomplish the required tasks. In other words, it is required that a dynamic construction of various related services and software be performed to meet a particular user's specific payment need. Many pieces of such software exist in the Web. The Web Services methods provide various supporting facilitates to describe the software for sharing and reuse. Among others, there are two issues significant for Micro-payment system development. The first is service description and selection of software components. Web Service Description Language (WSDL) and Universal Description, Discovery, and Integration (UDDI) are proposed standards for software component description. The purpose of the description is for the users to select the right and suitable software components. The second is service integration or composition of the selected software components. The integration issue is based on the service description. During the integration process, the service components, which fit together to serve the users requirements, are composed to form unified software for particular tasks.

Micro-payment is a software and service intensive activity, where many payment tasks involve different forms of services, such as payment method selection for buyers, security support software, product price comparison, etc. Using the web service technology will great enhance the quality and performance of micro-payment systems.

1.4 Paper Overview

The paper is organized as follows. In the next section, we describe some related work on both payment, micro-payment and security, as well as web services. Then, in section 3, we discuss general process and components of micro-payment procedure and system in the context of e-business so that we will learn where the micro-payment positions in the entire e-business picture. In section 4 we describe the architecture of micro-payment system with its components. Web service approaches to the micro-payment system are discussed in section 5, where we propose some business service patterns for micro-payment systems. Finally we conclude the paper in section 6 by proposing our future work in this direction.

2 Related Work

Not much research has been done for micro-payment related to web services, because researchers may have not realized its significance. However, a number of new concepts or terms have been turned up to response the needs of micro-payment, e.g. e-wallet, e-money, e-coin, etc.

A security issue for micro-payment has been discussed by [5]. There are many scenarios for transactions that involve amounts ranging from fractions of a cent to several dollars. For example, a customer may subscribe to a news service, paying for each article or picture. These transactions have too little values to require full protection of security. Although the risk of fraud involved in an individual transaction is small, however, the risk of large-scale fraud might be high. For example, an old fraud by programmers of banking records software was to save the fractions of a cent lost when the amount of daily interest on an account was truncated or rounded. Individually, the amounts were insignificant, but when taken over all accounts of a large bank for several years, they amounted to several million dollars.

In [4], Hauser, etc. considered micro-payments to be these payments, which are too small in amount to warrant the overhead costs of current financial clearing networks. The authors thought that an accumulation of small payments would be a regular payment, which could be handled as normal payment way. For the small payments, the authors proposed a third party, called micro-payment broker, which could be integrated into the Internet Keyed Payments Protocol (iKP).

The authors proposed two schemes for micro-payments in order to reduce the number of public-key operations required per payment. In the schemes, the players are brokers, users, and vendors. Brokers authorize users to make micro-payments to vendors, and redeem the payments collected by the vendors. While user-vendor relationships are transient, broker-user and broker-vendor relationships are long term.

Web Services are a very general model for building applications. Web Services can be implemented for any operating system that supports communication over the Internet. Web Services use the best of component-based development and the Web. Based on the idea of the best use of the services and software, Bennett, etc. propose a service-based model of software [1, 2], where services are configured to meet a specific set of requirements. The strategy of the approach is to enable users to create, compose, and assemble a service by bringing together a number of service suppliers to meet the needs of the end users.

3 Micro-Payment in the Context of E-business

Payment is an important aspect of the modern economy, and there are several payment mechanisms, which are well established for many years: cash, checks and charge cards among the most important. Charge cards are the common means of payments by phone, since the semi-secret card number can be passed to the seller over the phone (while physical cash and checks cannot be passed by phone). However, in the last few years, many new schemes have been presented, and quite a few were implemented or are being implemented. There were at least three different motivations for introducing new schemes, rather than passing credit card information as with phone orders:

- Security: the Internet poses increased threats to the use of payment mechanisms based on secret codes, such as charge cards. Users are concerned that their secret number would be revealed and abused. This is addressed by protocols, which secure payments while hiding the charge card number on transit (e.g. SSL) and even from the sell.
- Anonymity: users are often concerned about the privacy of their purchasing history (frequently associated with payments by cash). Electronic payments allow solutions to preserve anonymity, usually employing cryptographic techniques called anonymous cash. Such techniques were implemented for Internet-based payments, e.g. DigiCash, an electronic cash developed by eCash Technologies, Inc., see www.digicash.com.
- Minimize overhead and costs: credit cards involve a substantial minimal fee, about 2%, with minimal fees of about 25 cents. Furthermore, the authorization process involves substantial delay (due to the communication to the issuing bank, through the credit card network). These are significant problems for charging small amounts (cents) for information, services etc.

Micro-payment is an important complement to current e-payment methods. In e-business applications, e-payment is an important concept. Here we try to explain in the context of an e-procurement system. An e-procurement system usually consists of the following components:

- An online cataloguing component is to manage to select products required. When customers want shopping on the Internet, they hopes to know where they can find goods or items they need. In large supermarket, item labels are usually hanging above the goods. Similarly, the online cataloguing system helps the Internet customers to find items, in an even broad area and a convenient way because they can use search engines.
- An auction component is to support negotiation with the product suppliers or vendors. Negotiation is quite normal when people buy things. Customers always like to buy products with good quality and low price. The auction system enables customers not only to compare items in a wider range, but also to discuss with the suppliers about the prices, qualities, etc.
- An e-purchase system is to place order to the suppliers or vendors. When an order is ready to send to a product supplier, the pre-purchase process is over. Security protection is to start. The purchase order may need encrypted to prevent from distortion and guise. It may be necessary that some or all the items in the order need to be encrypted.
- An e-payment component is to check the delivery and to make payment to the suppliers. When the product arrives, the e-payment system starts to check the product against the original order. Then the payment is issued. An e-payment system, in general, involves at least three parties: a payment issuer, a payment receiver, and a certificate authority. Here the certificate authority provides authen-

tication to both sides of the trade. Banks or financial organizations usually play role of certificate authorities and authentication.

- In comparison with an e-procurement system, a micro-payment system may contain fewer components. It does not need e-auction, e-purchase, etc. In general, it requires low security protection but high response time because the customers just spend a small amount which they may not care about. However, waiting for 5 to 10 minutes to get what they buy would be quite annoying. "Click and buy" means that a customer just need to click a button and immediately gets what he buys. Of course, a billing record should also be presented to the customers when they ask.

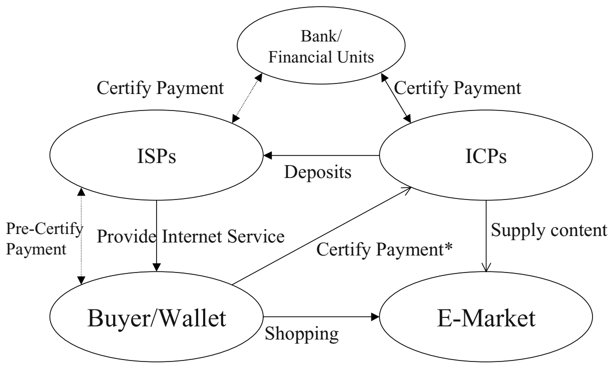


Fig. 1. Transaction and payment flow in a micro-payment system with different parties involved in the system.

4 System Architecture and Major Components

Let us imagine how micro-payment happens and proceeds, see Fig. 1. We assume that all transactions take place in the Internet. Similar to ordinary market, where people visit, select, and buy things, now we have an electronic market (e-market). You can enter the e-market, choose what you want and pay for them. Consequently, there should at least two basic components. One is a collection of goods (probably services or information) in the e-market that you can buy. The other is a wallet holding some money (usually small amount) enabling you to pay for your purchase. This virtual wallet is called electronic wallet or e-wallet. As a matter of fact, there are another three parties involved in the e-market and micro-payment. The first one is Internet Service Providers (ISPs), which enable you to connect to the Internet, like telecom companies. The second one is Internet Content Providers (ICPs), which provide products, goods, services, or other low cost items. The third one is a bank or a financial unit to certify the wallet you hold has a certain amount of electronic cash (money).

A micro-payment system consists of these components: e-wallet, billing system at ISP/bank, billing system at merchant. In the following, we describe these components and their connections.

The e-wallet component itself has three parts: an Internet connection to ISP and ICP, a small billing mechanism, and an interactive interface. The Internet connection allows it to access to the required certificate data from ISP or a bank (of course, ISP has already provided the user the Internet services) when necessary, and to ICP for shopping. The small billing mechanism (compared with the large billing system maintained at ISP side) lets the user know how much he has spent and how much left. The mechanism also provides a list of items the user has bought and other information. The interactive interface allows the user to enter general information, for example, login messages.

The ISP side billing system needs to perform three tasks. First, it should provide (if it has the authority) certificate of valid e-wallet to ICP or transfers such certificate from a bank to ICP. Second, it maintains all the users' billing records. Third, it periodically checks the customers' e-wallets and billing records.

The ICP side billing system performs two tasks. First, it maintains its own billing records for different customers. So it can report and get redeemed from the ISP. Second, it manages all the items to sell.

4.1 Transaction Flows

Here we attempt to describe a collection of possible certificate transactions among the micro-payment parties, see Fig. 2. First of all, we assume that the buyer's wallet, which has been initially certified by the ISP or bank, holds some e-money. This is called pre-certify payment. When a buyer has chosen some item to buy, by a click of payment, he issues a "certify payment" to ICP. Then ICP sends a message of "certify check" to ISP. This is the first round of certificate, from buyer, to ICP, to ISP.

Afterwards, ISP checks the ID and the amount the buyer has in his wallet. If the ID is correct and the amount is sufficient for this payment, the ISP sends a message "certify reply" back to ICP. This is the second round of certificate, from buyer, to ISP, to ICP. When the ICP gets "yes" from the ISP, it delivers the item to the buyer.

During the second round, the ISP, in this order, updates the billing records at the buyer's wallet, its own billing records, and the ICP's billing records.

If the buyer's ID is wrong or the amount is not sufficient, the ISP will send messages to both the buyer and the ICP.

4.2 Micro-Payment Infrastructure

In the micro-payment system, data transfer in the Internet is most important. Therefore, how to describe the transferring data so that such description can meet the current and future requirements of the Internet data exchange and maintenance is critical. Since W3C proposed XML (extensible markup language, see www.w3.org/XML) to

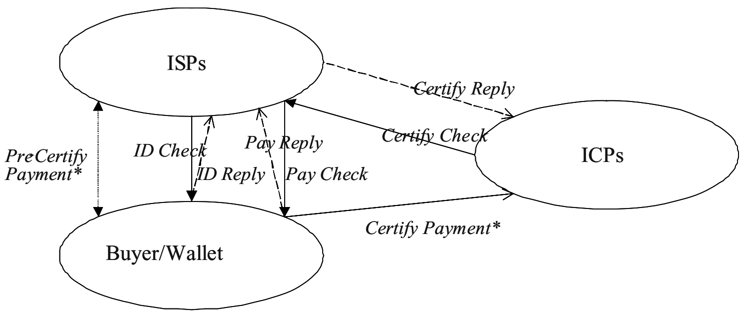


Fig. 2. Transaction flow among Micro-Payment components, including ISP, ICP, and e/Wallet.

be a standard for various data representation in the Web, XML has been widely accepted and used for description of a great variety of the Web resources. XML has the advantages of extensibility, separation of content and presentation, strict syntax, well-formedness, etc.

Recently, based on the XML specifications, a particular markup language specification for describing micro-payment is proposed to W3C, see www.w3.org. This specification provides an extensible way to embed in a Web page all the information necessary to initialize a micro-payment (amounts and currencies, payment systems, etc). This embedding allows different micro-payment electronic wallets to coexist in an interoperable manner.

This specification defines a set of tags for description of payment related information to be transferred on the Web.

5 Web Service Investigation for Micro-Payments

Micro-payment is a complex, software component based, distributed system. It involves different parties for different functions. To accomplish one micro-payment transaction, many services are required, such as setting up appropriate security channels, checking the payer's identity, making daily purchase records, and so on, see Fig. 3. These business services form different patterns [10], serving different users' requirements and purposes. In order to describe the businesses and services in a semantic rich manner, we need to use the current advanced semantic web technology, for example, RDF, to build up a semantic description framework for payment business services. This framework is also used to describe the end user's requirements and profile, for a better requirement match from service providers and service consumers. In the following we will further discuss these three aspects of services: business patterns, service description, and requirement matchmaking.

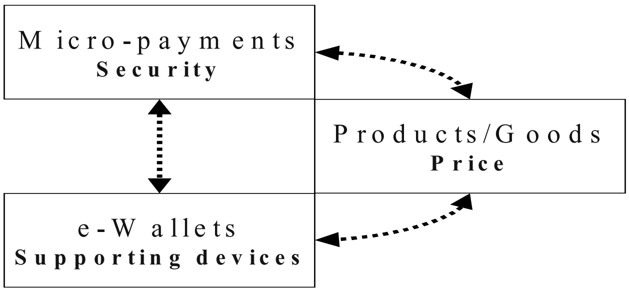


Fig. 3. MP Components play roles in a Web Service system, involving payment security services, product pricing services, and e-Wallet supporting services.

5.1 Service Patterns for Micro-Payment

Service patterns suggest some forms that services are organized to serve certain purposes or to accomplish certain tasks. Such form or model is peculiarly significant in the situations where many distributed services are linked, integrated, coordinated, and cooperated together with their data sources and even human power sources for achieving a tremendous target. These situations or organizations are called "virtual organization".

These service patterns can be considered to be in these three forms, a service integration pattern, a service aggregate pattern, and a service composite pattern. The service integration pattern maintains a set of services or software which is organized in parallel. When a service request comes, a "broker" checks the requirements of the request and finds a most suitable one to meet the request. To the users, it appears as if there were only one service, for example, payment service, that the users need to know. In fact, the users are using a number of payment services, for example pay-by-e-cash, pay-by-e-wallet, pay-by-e-card, or using payment services provided by different service providers.

The service composite pattern has been widely discussed and is mainly considered to be a line of services. Each service is dependent on its preceded service to provide inputs and supplies outputs to its follower. A special situation is that one service may request or call another service in order to fulfill its own tasks. The key issue here is dynamic composite of services with a secure or fault torrent mechanism.

The service aggregate pattern indicates the situations where a set of services available but one of them is selected according to the users' requirements. These services almost serve the same purposes and do the same functions. Their difference may lie in for example, low cost but light security support, better usability but less functionality, etc. This pattern emphasizes more on service availability whereas the composite pattern focuses on-demands.

How to develop these business patterns for payments is heavily dependent on a good semantic description method, which can provide rich semantics for services and better information exchange through the Web. The aim to propose these patterns is to suggest some business models of payments that better serve the users with web service techniques.

5.2 Service Description for Payment

According to the triangle architecture for web services [11], a web service involves three parties, a registry where the service is registered and "described", a service provider that also provides service specifications, a service consumer or user that makes service request together with sort of service description. Currently almost all of the service description is described in UDDI and stored in the registry. The service description mainly contains business description and service description. The business description is about the business situation of the service provider and the service description is about the services that the business is providing. However, The UDDI description of business and services need to be improved to possess richer semantics for better matching between the users requirements and the service description.

Payment service description. We attempt to maintain a semantic description framework to provide full description for a payment service. The framework will include a static part, which describes general transactions methods and general payment processes, and a dynamic part, which stipulates operations such as match bindings, etc. Following is an example of the semantic description framework (static part) for payment transaction.

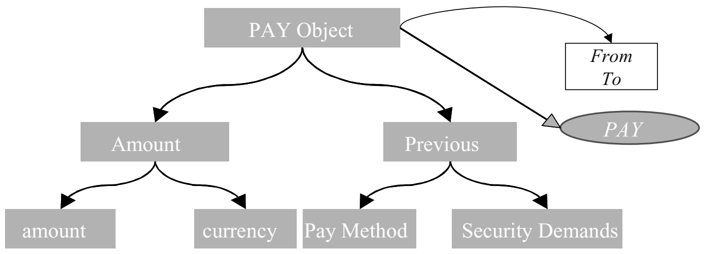


Fig. 4. The descriptive ontology for payment object in terms of the RDF standard aims to provide further web service based support.

A transaction object includes transaction ID, transaction description, transaction properties, and transaction ontology. Transaction ID is the unique number for the transaction, which is used to identify the transaction. Transaction Description is a short description for the transaction for human understanding and NL semantic processing. Transaction Properties are RDF based description for the transaction. The properties include, for example, name, relationships with other transactions, transaction types, etc. Transaction ontology describes a set of hierarchical structures, such as conceptual taxonomy, process hierarchy structure, etc.

In Fig. 4, we illustrate a payment object with its description elements. The description is based on RDF. The pay object is related to a number of values through the properties like "From" and "To". It is also related to a function property, called

"PAY". The object is about a "previous" payment, which includes the payment amount, payment methods and security requirements. The above description fragment is, however, our first attempt to formalize the description for payment transactions.

Service function description. Service function is currently not well very described part within service description but it is a useful part for improving the user requirement matches. As it is difficult for a business to provide a description of its services meeting some normalized form, they are in most cases written in natural languages with very informal structures. In our framework, we also propose a formal specification for service function description. This formal description mainly concerns that the services accomplish what tasks, perform what processes, possess what features, and involve what other processes and services. For example, the functions for a payment service include access checking, identity verification, accounting, clearance, etc.

User request description. As a most important party in business service, the users description will heavily influence the quality of requirement matching. However, this work is now still left to the end users. The end users have to input some keywords and select some ontology terms for an application domain to a service registry in order to search for the services they request. In most cases, the users do not know their exact requirements for services and even do not know how to structurally form their requirements. Their expression of requests can be very vague and ambiguous. To support the end users with their payment requests, for example, the users' requirements can be transferred in a certain structure together with their profiles.

Suppose that a user wants to download a single song from a Web-based music shop and pays 50 cents for it. The user may demand a payment service for this transaction using his "e-wallet" and to be "secure". His "e-wallet" provides e-cash, e-card, phone-bill account. By "secure" he means that a security channel should be set in SSL. According to these requirements, a payment service is searched against the UDDI registry and a matchmaker mechanism is performed to find a best match. Here we need to emphasize that to structure the users' requirements and to use a semantic framework to describe the users' profile is extremely important. The users' profile helps us with good understanding of "secure".

5.3 Service Matchmaking for Micro-Payment

Initially, the matchmaking process is to find the services from the service registry described in UDDI according to the users' requests. However, the users' requests are currently expressed only in terms of service names or keywords. Many approaches have been proposed to enrich the semantics for the requirement matchmaking [6, 8, 12]. The main feature of the approaches is to introduce ontological structure for description of services and businesses. For example, in order to provide more semantic support to the description and search of services in the UDDI based registry, a set of mapping rules can be established between a semantic framework and the UDDI registry, where each service description item is mapped into a registry entry. Then richer semantic description for businesses and services, as well as the users requirements can be achieved using the semantic framework.

Our semantic description framework model, based on the W3C recommendation – RDF and RDFS, contains the following components: a characteristic based semantic description modeling component for representing the businesses and general services; a process based description modeling component for expressing service functions and processes; a structured requirement modeling component to represent the end users' requirements and requests, as well as their profiles. In addition, we also use an ontology model for representing the business ontology and service ontology, where the domain knowledge is structurally expressed.

The semantic description is useful also in matching techniques, which are used for finding required services or selecting a best match among a number of service candidates. Currently, we are investigating a formal match method, which we call graph match. Suppose that *a* and *b* are two service nodes with an edge *ab* indicating a possible relationship, say, *a* depending-on *b*, between them and *G* is a service description schema. There may be a sub-graph *G'* in G which contains nodes *a* and *b*, as well as a chain of edges linking *a* and *b*, which contains *ab*. With support from the semantic relativity approach [9] and large number of services available, this graph match technique will be more effective for selection of best services.

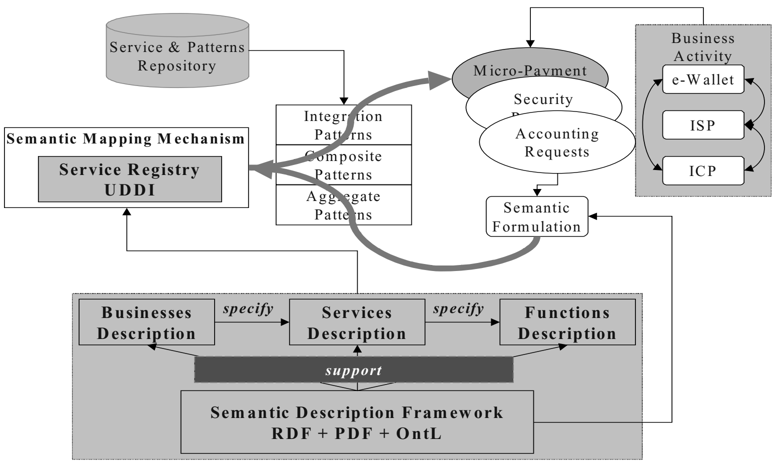


Fig. 5. The integrated architecture contains three major components: a semantic descriptor, a pattern constructor, and a user request formation

5.4 An Integrated Architecture for Services

The integrated architecture contains three major components: a semantic descriptor, a pattern constructor, and a user request formation, see Fig. 5. All these components are based on the semantic description framework. The semantic descriptor is for business, service, and function description. As we discussed earlier, business description specifies business services and services specifies service functions. In addition, ontology support for general business and service is also maintained within component.

The pattern constructor uses the service and pattern repository to collect the matched services discovered through matchmaking and semantic mapping mechanism to form the three services patterns, integration, aggregate, and composite. For the integration pattern, the service function description will be critical because to couple different services together to form a chain of services for a specific purpose requires accurate function specifications.

The user request formation is to apply the criteria supplied from the semantic description framework and the ontology structure to re-structure the user requests, so that the requests will bring more semantics for later matchmaking with the service description within the registry component. Currently we are developing a graph matching algorithm, in which a user request is seen as a sub-graph and the service description schema is considered to be a big service graph [8].

6 Conclusion

In this paper, we have generally introduced an analysis of the development of micro-payment, discussed some suitable security protection methods for micro-payment, and described some business and service patterns for current micro-payment systems. We believe that with gradual deployment of web service techniques, micro-payment will gain larger market than before because cheap, flexible, distributed services will great support the payment chains.

Although we have discussed the use of semantic description framework for micro-payment processes and architecture, we still consider that this will be a long procedure because firstly it is not easy to build up a suitable semantic rich application to cope with the inherent problems, and secondly current service description methods based on UDDI and WSDL may hinder the progress of adopting semantic richer methods.

Matchmaking is a critical issue, which deals with how suitable a service can be selected to meet the real requirements. Currently there are algorithms for making matches between the users' requirements and the description of the services sought. However, the users' requirements are expressed mainly using keywords. This approach greatly limits the semantic expressiveness of the users' requirements

Our current research work is to develop a usable semantic framework for web services as well as for the users' demands, so that matchmaking can be carried on in a semantic richer description framework. We will fulfill the integrated architecture for micro-payment service description. An experiment is ongoing to test the semantic description framework of services for micro-payment.

References

1. K. H. Bennett, N. E. Gold, P. J. Layzell, F. Zhu, O. P. Brereton, D. Budgen, J. Keane, I. Kotsiopoulos, M. Turner, J. Xu, O. Almilaji, J. C. Chen and A. Owrak, A Broker Architecture for Integrating data Using a Web Services Environment, in Proceedings of 1st International Conference on Service-Oriented Computing, 15-18 December 2003, Trento, Italy
2. K. H. Bennett, N. E. Gold, M. Munro, J. Xu, P. J. Layzell, D. Budgen, O. P. Brereton and N. Mehandjiev, Prototype Implementations of an Architectural Model for Service-Based Flexible Software in Proceedings Thirty-Fifth Hawaii International Conference on System Sciences, edited by Ralph H. Sprague, Jr., 2002
3. E. Cerami, Web Services Essentials – Distributed Applications with XML-RPC, SOAP, UDDI & WSDL, The O'Reilly Network, February 2002.
4. R. Hauser, M. Steiner, and M. Waidner, Micro-Payments Based on iKP, in Proceedings of Fifth International World Wide Web Conference (WWW5), May 6-10, 1996 - Paris, France.
5. A. Herzberg and H. Yochai, Mini-Pay: Charging per Click on the Web, in Proceedings of the sixth international conference on WWW, 1997
6. M. Paolucci, T. Kawamura, T. R. Payne, K. Sycara, Semantic Matching of Web Services Capabilities in Proceedings of First International Semantic Web Conference, Sardinia, Italy, Editors: I. Horrocks, J. Hendler (Eds.) June 9-12, 2002.
7. R. Rivest and A. Shamir, PayWord and MicroMint: Two simple micropayment schemes, in Proceedings of Fifth International World Wide Web Conference (WWW5), May 6-10, 1996 - Paris, France.
8. W. Song, Semantic Issues in the Grid computing and Web Services, in the Proceedings of International Conference on Management of e-Commerce and e-Government, Nanchang, China, Oct. 2003
9. W. Song, Semantic Object Relativity in Schema Integration, in Proceedings of CAiSE, Utrecht, The Netherlands, 1994
10. G. Vasudeva, Patterns for business, IBM White Paper, 2003.
11. L. Zhang, Q. Zhou, and J-Y. Chung, Developing Grid computing applications, IBM Report, IBM T.J. Watson Research Center, New York, USA.
12. Y. Zhang, W. Song, K. Bennett, Semantic Enrichment for Web Service Description and Matchmaking, Department Report, Computer Science, University of Durham, UK. 2004

Implementation of a Service Oriented Architecture at Deutsche Post MAIL

Michael Herr, Uwe Bath, and Arne Koschel

Deutsche Post AG
53113 Bonn, Germany
{Michael.Herr, Uwe.Bath, Arne.Koschel}@deutschepost.de

Abstract. This paper explains the background and some of our experiences in implementing an integration infrastructure. It describes representative problems of historically grown IT landscapes and shows experience how Deutsche Post MAIL solved this challenge. For that purpose we introduced a business-driven enterprise application architecture and point out its main features and benefits. Afterwards we give an overview on Service Backbone as the technical implementation of the corresponding integration infrastructure. We address IT managers, business architects as well as IT professionals, who want to understand which problems can be solved by establishing a service oriented architecture and how Service Backbone can be used as an integration infrastructure within a SOA [1].

Keywords: Enterprise Applications Architecture (EAA), Service, Web Services (WS), Domain, Service Backbone (SBB), Service Oriented Architecture (SOA), Integration Infrastructure

1 Initial Situation at Deutsche Post MAIL

1.1 Problems and Motivation

Large companies like Deutsche Post usually suffer from their historically grown IT landscape, which often results in severe application related challenges as well as data related problems.

Typical application related shortcomings are weak support for core business processes, narrowly focused island solutions and lack of understanding for holistic, business-driven approaches. Furthermore applications are characterized by no clear functional boundaries. This leads to high functional redundancy between applications and high effort for modifications.

Moreover a historically grown IT architecture is usually characterized by the poor availability of the core information (revenues, cost, competitor information, etc.) needed today and in future. This results in fragmented responsibilities and fragmented data ownership, complex and only partially defined data consolidation procedures and no consistent modelling of commonly used central data. Gartner Group describes such a typical existing IT integration landscape vividly as “IT integration spaghetti” (see Figure 1).

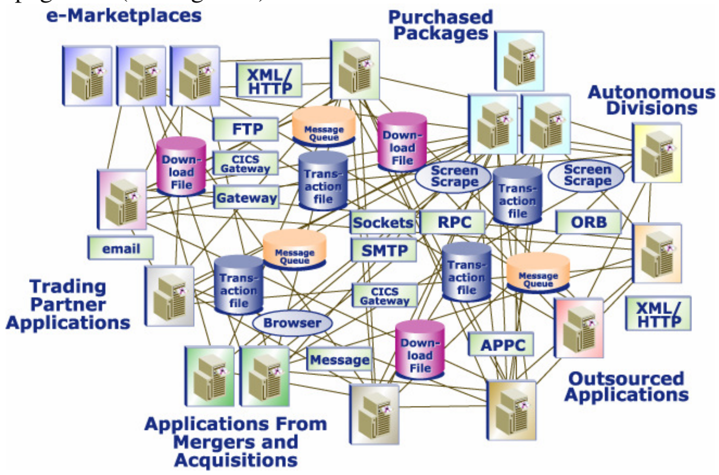


Fig. 1. IT “Integration Spaghetti” (Gartner Group)

The administration and maintenance of such historically grown IT landscapes cost a considerable amount of the entire IT budget, which is then missing for important new developments to remain competitive [2].

1.2 Our Strategy: Develop a Modular Enterprise Application Architecture for Integration

The key approach to disentangle the complex application landscape with all the shortcomings mentioned above is based on the idea of a business–driven Enterprise Application Architecture (EAA) which abstracts the IT infrastructure.

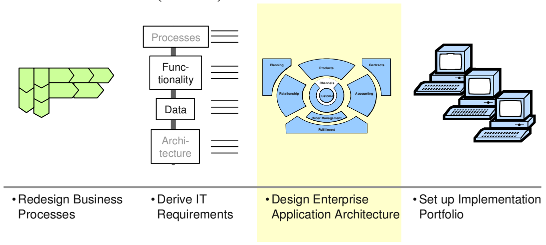

Processes
Func-tionality
Data
Archi-tecture
• Redesign Business Processes
• Derive IT Requirements
• Design Enterprise Application Architecture
• Set up Implementation Portfolio

Fig. 2. From business processes to IT applications using enterprise application architecture

Initially, all relevant business requirements were analyzed. In many cases this involved a complete redesign of business processes. For each business process all input-output relations were examined and IT requirements, data and functionalities were derived. This formed the basis of our EAA. It consists of modular components by centralizing closely related functionalities and data to domains, which are extracted from the redesigned business processes as described before.

Domains provide their data and functionality by stable defined interfaces, called "services". Therefore our EAA refers to a Service Oriented Architecture (SOA) [3,4,12,13,14]. This leads to the concept of services, which are made available over an enterprise service bus and can be combined for new applications.

This idea, like much else in the software industry, is not an entirely new one, but rather an evolutionary development based on long-established principles. Some years ago, the buzzword was component-based development; today, we encounter an enhanced version of this idea in the form of SOA. Also not far ago CORBA-based [5] "service bus" integration infrastructures were successfully deployed [6] (and are in production use), e.g. in 1999 at companies like Credit Suisse [7] or Zurich Financial Services [8]. Nowadays SOA projects are very often build based on Java/J2EE [9,10], XML [11] & Web Services technology [3,12,13,14], e.g. at the Sparkassen-Informatik-Zentrum [15] or at EuroHyp [16]. However, we believe, that the EAA and our Service Backbone (SBB), which we will describe below, belongs to the most flexible approaches w.r.t service distribution, message flow and processing etc.

1.3 Conclusion

The business drives the services, and the services drive the technology. In essence, services act as a layer of abstraction between the business and the technology. Building new IT applications therefore requires a clear understanding of the dynamic relationships between the needs of the business and the available services. The next chapter attends to give more details and important features about it. Collecting specific business requirements and mapping them to the EAA allows the development and evaluation of reasonable implementation alternatives. This leads to precise designs of IT applications and to a well-defined application landscape.

2 Characteristics of an Enterprise Application Architecture

In general, an EAA provides a business-driven view on IT applications, their interfaces and their interconnections. The concept of our EAA can be compared to a development plan of a city.

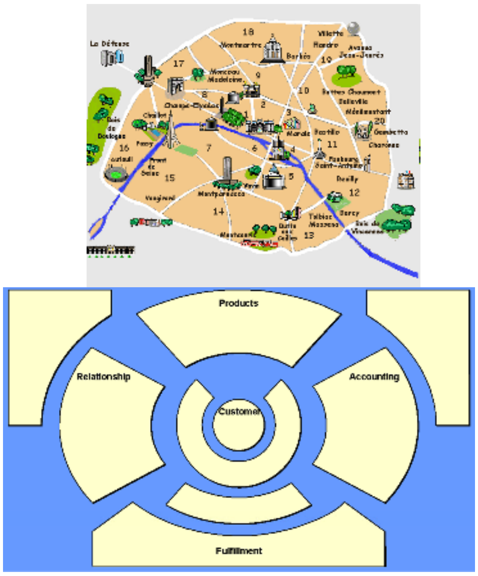


Fig. 3. Analogies between a development plan of a city and an enterprise application architecture

It describes reasonably separated components, called domains, containing the main business logic. These domains can be compared to different areas of a city, which have all clearly defined properties (like residential area, industrial area, airport, parks, shopping centre etc.).

The access to the business logic encapsulated by domains is realized by services. In our analogy to cities this refers to their infrastructural components like streets, rivers, rails, electricity and water supply, which connect the different districts to each other. The establishment of standards for all infrastructural components is the key to both architecture approaches' success.

2.1 Features and Benefits of Domains

Domains are the fundamental element of the EAA. They consist of modular defined functionality, which is necessary to support business processes and the underlying basic data. Within dedicated IT projects, these requirements have to be implemented.

The main characteristics of domains are:

- Domains encapsulate their functionality and data.
- Functionality is implemented redundancy-free, information is consistent.
- Functionality and data can be used everywhere, they can be combined to support ever new business processes.
- New projects can build upon existing assets, investments are secured.

2.2 Features and Benefits of Services

In a SOA services are the only way to get access to the functionality and data of a domain. The main characteristics of services are:

- Services are the interface to the business logic encapsulated by domains.
- Each service will be provided by a dedicated application ("service provider").
- Services should not be tailored to one specific service consumer. Instead, they should be general enough to be used by multiple service consumers ("coarse-grained"). This results in increased efficiency and cost savings.
- Services have a long-term stability. They describe main business interactions, which are usually much more stable than the implementation of specific business requirements.
- A service separates the service provider from its service consumers, which allows fast and independent changes of their implementations (as long as the service is kept stable). A new service version can be introduced, if changes according to the service interface are necessary.

3 Our Comprehension of a Service Oriented Architecture

Deutsche Post MAIL decided to implement its enterprise application architecture with a service oriented architecture). Conceptually the SOA community defines three main participants: service consumer/requestor, service provider and service registry/broker. At a technical layer they describe the need for an integration middleware to support features such as transport, routing or limited transformation.

As the main concept of SOA, a service encapsulates in our understanding a defined unit of business logic, which is reusable. It hides its implementation and can be invoked by a service consumer. Semantics, syntax and quality of service – on basis of the business logic they represent – are the main elements that describe a service. WSDL is an important standard to describe services but not sufficient to describe semantics and quality of service.

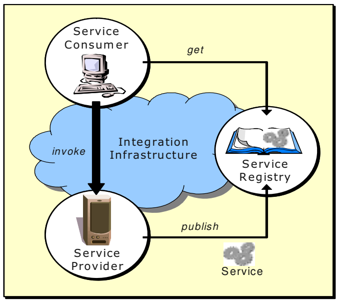


Fig. 4. Participants in a service oriented architecture (overview)

In accordance with the SOA community the participants – service consumer, service provider and service registry – play a key role in our SOA understanding:

- A service provider implements and provides a specific service, which must be published in a service registry. The published service is described by semantics, syntax and quality of service.
- Service consumers can invoke this service after looking it up in the service registry.
- The service registry enables the publishing of services and their discovery at design and run-time.

The service consumer and service provider are business service participants. They are responsible for implementing and supporting business logic. They are embedded in a decentralized and loosely-coupled way, in line with the integrated service paradigm.

Besides the described participants our SOA understanding includes a fourth conceptual participant: the integration infrastructure.

The integration infrastructure enables communication between service consumer, provider and registry.

- It is responsible for transport of messages through a given network infrastructure.
- It provides authentication and authorization.
- It enables monitoring of service consumption and contracts.

The integration infrastructure aims at facilitating business service participant communication. New service participants can connect easily with each other. The integration infrastructure undertakes the task of generic and common functions to support business service participants.

This support for business service participants comprises functionalities such as

- Message transport including persistent messaging and compression
- Enabling logical addressing to invoke a service (dynamic binding)
- Message validation and transformation
- Authentication and authorization
- Service management including system management, life cycle management and business management

The integration infrastructure itself is normally also implemented service oriented and composed of technical service participants. These technical service participants enable functionality (e.g. authentication and authorization, transformation), that the integration infrastructure provides to the business service participants.

Step-by-step Service Backbone implements our vision of a service oriented integration infrastructure as described above. Technical service participants (e.g. single sign-on, transformation, user & rights) are additionally provided as "technical services" by the infrastructure, which supports the flexibility of our best-of-breed solution.

4 SBB: The Backbone of Our Service Oriented Architecture

A SOA needs a capable integration infrastructure technology to support communication between all service participants. Therefore Deutsche Post MAIL built the Service Backbone (SBB), which became operational in December 2001.

SBB represents the technical implementation of our integration infrastructure of a SOA. The main objective of Service Backbone is to receive service calls from service consumers and to transport these calls to dedicated service providers and vice versa.

4.1 Key Features of Service Backbone

SBB realizes a comprehensive integration infrastructure. The components are strictly developed in a standardized and adaptable way, which is to date not available out-of-the-box.

The key features of our implementation of Service Backbone are:

- Easy-to-use interface to connect technically to SBB; including several adapters to legacy systems, partially off-the-shelf
- Comprehensive directory for all services available using Service Backbone
- Syntax and type validation of all documents transported by Service Backbone, can be performed on service consumer, service provider or on both
- Transportation mechanisms for different interaction styles (synchronous, asynchronous, one-way, publish/subscribe); including data compression
- Exhaustive user directory used for authentication and authorization
- Transformation engines for structural document mappings between XML schemes as well as content matching

4.2 Design Choices for Service Backbone

SBB uses IT standards whenever it’s reasonable and possible in order to avoid the risk of proprietary tool vendor lock-in, which would ultimately lead to an "EAI legacy" problem. The infrastructure provides a holistic, working solution by integrating open, best-of-breed tools and makes sure that integration infrastructure is proven and managed. This ensures tool and vendor flexibility.

SBB supports well-known and proved third-party-products. At present, these are IBM MQSeries (via JMS), BEA WebLogic (based on J2EE 1.3), Sun ONE Directory Server (via JNDI and LDAP) and Apache Tomcat.

4.3 Architecture of Service Backbone

SBB is composed of three main components “Local SBB interface”, “Central infrastructure components” and “Technical Service Participants”, which are explained in more detail in the following sections.

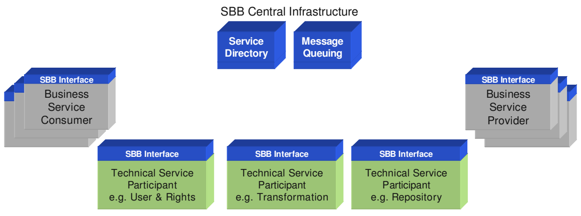

SBB Central Infrastructure
Service Directory
Message Queuing
SBB Interface
Business Service Consumer
SBB Interface
Business Service Provider
SBB Interface
Technical Service Participant e.g. User & Rights
SBB Interface
Technical Service Participant e.g. Transformation
SBB Interface
Technical Service Participant e.g. Repository

Fig. 5. Components of SBB

To connect service participants, each service participant has its local SBB interface attached. The Service Programming Interface (SPI) is a local SBB interface for service provider, and the Application Programming Interface (API) is a local SBB Interface for service consumer.

A developer calls a service by invoking the API and passing an XML document (message) specifying the input parameters. SBB carries this document to the provider's SPI which invokes the requested operation. To develop with the local SBB interface only Java, XML and XML Schema skills are required.

4.4 Central Infrastructure Components

There are only two central infrastructure components for handling service calls.

Service Directory: It contains all required information (e.g. binding information, users & rights) to enable security, dynamic binding and versioning. Technically this is based on an LDAP server and a web server. Any available LDAP server and web server are applicable.

Message Queuing: The Message Queuing is used for persistent and normally for asynchronous service calls. Every Java Message Service – (JMS) oriented Message Queuing is applicable.

Of course replication and clustering of these infrastructure components is possible and due to high availability reasons it is recommended.

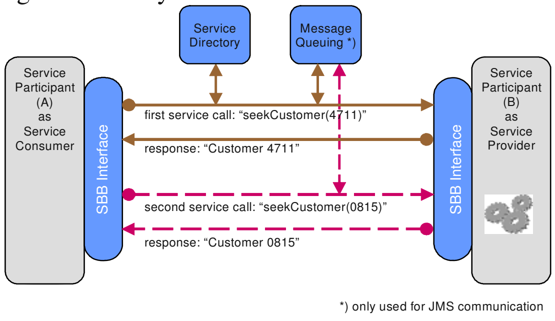

Service Directory
Message Queuing *)
Service Participant (A) as Service Consumer
SBB Interface
first service call: "seekCustomer(4711)"
response: "Customer 4711"
second service call: "seekCustomer(0815)"
response: "Customer 0815"
SBB Interface
Service Participant (B) as Service Provider
*) only used for JMS communication

Fig. 6. Service call using SBB central components

When Service Participant (A) tries to call Service Participant (B) for the first time, the central SBB infrastructure components will be used. In so doing, the necessary call information will be stored locally in the SBB interface of related participants. A second service call does not mandatory need to use all SBB infrastructure components again, as the connection is established directly between the participants via the SBB interface. The process of determining the refresh rate for verification of the connection information is user-friendly manageable.

4.5 Technical Service Participants

A technical service participant is responsible for carrying out a defined integration infrastructure functionality and supports the local SBB interface (API and/or SPI).

Currently SBB includes the following technical service participants:

- Transformation (Matching)
  A common Integration Broker called "Integrated Transformation Service (ITS)" provides complex content transformation.
- Service Directory Administration
  For administration of the service directory there is a set of tools which are developed as technical service participants. These tools allow deployment of services and administration of users and rights. They are built as J2EE Web Application so every available servlet engine is applicable.

It is planned to support further technical service participants like Single Sign-On (pilot project), Data Integration and Users & Rights.

4.6 Service Backbone and Web Services

Web services are increasingly becoming an adequate technology for the partial implementation of features of a service oriented architecture.

What are Web Services? A Web service is a software designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [11].

SBB relies on the same service concept as web services and combines web services technology, like SOAP-messages and HTTP, with reliability messaging due to an underlying MOM (JMS). As an add-on SBB supports SOA features that are not yet addressed by web services, for instance security, dynamic binding, versioning and reliability.

We call a service an "SBB Service" if it is connected to and reachable via SBB as integration infrastructure. As a result of supporting web services enhancing features such as versioning and security SBB Service requests and responses are enriched compared to web services. We intend to support the publication of SBB Services as web services (based on WS-I Basic Profile 1.0).

4.7 Benefits of Our SOA and SBB

4.7.1 Reduction of Interface Complexity

The power and flexibility that SOA potentially offers to the enterprise are substantial. If an organization abstracts its IT infrastructure so that it presents its functionality in the form of services that offer clear business value, the overall interface complexity can be reduced dramatically.

4.7.2 Decentralized Software Development

Individual service implementation projects can be carried out flexibly and decentralized. This avoids the risks inherent in large projects and allows the application landscape to be developed step by step while maintaining the strategy flexibility in the long-term. At this the business driven Enterprise Application Architecture supports the coordination of the individual developments and helps to avoid uncontrolled growth.

4.7.3 Explicit Separation of Business Logic and Service Mediation Logic

In a SOA the complete business logic is encapsulated in domains. The integration infrastructure Service Backbone is only responsible for the service mediation and, if necessary, data transformation. Classical EAI tools usually prefer a hub-and-spoke approach for integration, which requires a part of the business logic within the integration tool. This may lead again to complex dependencies between business applications and the integration infrastructure and may also introduce a performance and scalability bottleneck, which should be avoided.

4.7.4 Technical Independency of Service Participants

Service consumers can access services independently of the underlying technology that supports them. If service consumers can discover and bind to available services, then the IT infrastructure behind those services can offer extraordinary flexibility to the businesses that invoke them.

5 Conclusion: Experiences and Recommendations Operating a SOA

After more than three years operating SBB successfully we gained a lot of experience in implementing a SOA. The following statements summarize these experiences in a brief conclusion.

1. If you want to reduce IT complexity through integration, start with the business (logical) view and use an architectural approach.

2. Focus on services (multiple use of data and functionality), not on data interchange (point-to-point communication).
3. Don't neglect the integration infrastructure layer - this is much more than just "data transport".
4. The technical integration infrastructure is characterized by long-term stability, so better stay vendor independent and stick to open standards.
5. The success of a whole integration project mainly depends on the acceptance of all business-driven ser-vice participants. So don't let the IT professionals drive the effort alone.

Armed with the necessary commitment of all persons involved, we have been approaching our aims step-by-step. Structuring an integrated service oriented architecture – SBB itself is also an instance of a SOA – was not very easy. But up to now, with our decision to orient ourselves to open standards – avoiding vendor lock-in and following the strategy of loosely coupled components - we have been successful in application integration. The benefit of using services in a multiple way is now increasingly seen in Deutsche Post MAIL and in the SOA community. Our next strategic objective will involve confronting a difficult task: process integration and collaboration.

Abbreviations

EAA	Enterprise Application Architecture
ITS	Integrated Transformation Service
J2EE	Java 2 Enterprise Edition
MOM	Message Oriented Middleware
SBB	Service Backbone
SOA	Service oriented architecture
WS	Web Service
WS-I	Web Service Interoperability
XML	eXtensible Markup Language

Acknowledgements.
We would like to thank all friends and colleagues at Deutsche Post for their support in transforming the vision of the Service Backbone into a reality.

References

[1] „Geschäftsprozesse bestimmen die Architektur: EAI senkt IT-Risiko der Post", Jan Schulze, Computerwoche Nr. 28 vom 12.07.2002.
[2] „Keep IT simple!", Interview mit Dr. Johannes Helbig, CIO des Unternehmensbereichs BRIEF, Management Mail 4/2003.

[3] „Portal for SOA and Web Services“, http://www.service-architecture.com.
[4] „What You Need to Know About Service-Oriented Architecture", Todd Datz,, Jan. 2004, CIO Magazine, http://www.cio.com/archive/011504/soa.html.
[5] „Common Object Request Broker Architecture: CORBA”, Object Management Group (OMG), http://www.corba.org.
[6] „CORBA success stories”, Object Management Group (OMG) http://www.corba.org/success.html.
[7] „Credit Suisse Information Bus Case Study. http://www.iona.com/info/aboutus/ customers/creditsuisse.html.
[8] „Mainframe Integration Case Study.”, Zurich Insurance. http://www.iona.com/info/aboutus/customers/casestudies/zurich.pdf.
[9] „Java”, Sun, http://java.sun.com/j2ee.
[10] „Java 2 Enterprise Edition: J2EE”, Sun, http://java.sun.com/j2ee.
[11] „World Wide Web Consortium”, http://www.w3c.org.
[12] „Web Services”, http://www.webservices.org.
[13] „Web Services Special”. Informatik Spektrum. Springer. April 2004.
[14] „Web Services zur Integration“, A. Koschel, R.Tritsch JavaSPEKTRUM 3/4 2002.
[15] „Web services-oriented architecture in production in the finance industry”. M. Brandner, M. Craes, F. Oellermann, O. Zimmermann. GI Informatik Spektrum. Springer, April 2004.
[16] „Flexible Value Structures in Banking”, U. Hohmann, M. Rill, A. Wimmer, Communications of the ACM, 47(5), May 2004.

Web Services for Integrated Management: A Case Study

Jean-Philippe Martin-Flatin[1], Pierre-Alain Doffoel[2], and Mario Jeckle[3]

[1] CERN, Geneva, Switzerland
jp.martin-flatin@ieee.org
[2] ESCP-EAP, Paris, France
pa@doffoel.com
[3] University of Applied Sciences Furtwangen, Furtwangen im Schwarzwald, Germany
mario@jeckle.de

Abstract. As evidenced by discussions in standards organizations, vendors and the user community have recently showed a growing interest in using XML technologies for management purposes. To investigate the relevance of this approach, we have added support for Web Services to JAMAP (a Java-based research prototype of a management platform) and managed a gigabit transoceanic testbed. In this paper, we present the main lessons learned during this process and attempt to draw conclusions of general interest as to the applicability of Web Services for managing IP networks and systems. Our main conclusions are that XML, WSDL and SOAP are useful, especially for configuration management, whereas UDDI is not adequate. To date, we still lack a standard way of publishing, discovering and subscribing to Web Services for the purpose of managing network devices and systems.

1 Introduction

Recent discussions in the Internet Engineering Task Force (IETF), Internet Research Task Force (IRTF) and Distributed Management Task Force (DMTF) demonstrate the management community's growing interest in using Web technologies—especially those based on the Extensible Markup Language (XML)—in management systems [16]. At the same time, the consortia in charge of standardizing Web Services technologies have showed an increasing interest in management.

In the DataTAG Project [7], we wanted to manage gigabit network devices and systems in a distributed manner. We had a cluster of PC servers and network devices (routers and switches from different vendors) at both ends of our transoceanic gigabit network (at CERN in Geneva, Switzerland and at StarLight in Chicago, IL, USA). To assess the suitability of Web Services for the purpose of management, we ported JAMAP [11], an open-source research prototype of a management platform, to Web Services. We could have used integrated and more sophisticated software suites such as IBM's WebSphere and BEA's WebLogic, or a management product such as HP's Web Services Management Framework. But using JAMAP gave us complete control over the code and allowed us to change each component independently of the others.

The remainder of this article is organized as follows. First, we review the main building blocks of Web Services. Next, we summarize the management architecture that underpins JAMAP. Then we analyze the advantages of using XML-based technologies in integrated management. Next, we describe how we leveraged Web Services in JAMAP and draw some lessons of general interest. Finally, we conclude and present directions for future work.

2 Overview of Web Services

Because interoperability is crucial to Web Services, their standardization has been of key importance since their inception. So far, two consortia have been particularly active in this field: the World Wide Web Consortium (W3C) [25] and the Organization for the Advancement of Structured Information Standards (OASIS) [15]. More recently, a new industrial consortium, the Web Services Interoperability Organization (WS-I) [24], has begun standardizing interoperability aspects of Web Services.

2.1 W3C Activities

The W3C's activities on Web Services are split into four areas:

– Simple Object Access Protocol (SOAP)[1]
– Web Services Definition Language (WSDL)
– Web Services Architecture (WSA)
– Web Services Choreography (WS-Choreography)

SOAP [8][9] is a protocol that allows applications to exchange structured information in a distributed environment. The SOAP binding used by most applications specifies how to carry a SOAP message within an HTTP entity-body; with this binding, SOAP can be viewed as a way to structure XML data in an HTTP pipe between two applications running on distant machines. A SOAP message consists of a header and a body. The header is optional and carries metadata; the body is mandatory and includes the actual application payload. A SOAP message is used for one-way transmission between a SOAP sender and a SOAP receiver, possibly via SOAP intermediaries. Multiple SOAP messages can be combined by applications to support more complex interaction patterns such as request-response. The SOAP specification also specifies an XML representation for Remote Procedure Call (RPC) invocations and responses carried in SOAP messages.

WSDL [6][10] is an XML language for describing Web Services. Building on the base communication mechanism defined by SOAP, each Web Service is characterized by its signature, which consists of a procedure name, the type of the result returned by this procedure, the names and types of the procedure parameters, and the actual message pattern chosen for communication. Multiple Web Services can be published in a single WSDL file, often called a WSDL repository. If we compare Web Services with CORBA, the WSDL language is similar to the Interface Definition Language (IDL); a WSDL repository is similar to CORBA's Interface Repository; Web applications may discover Web Services in a WSDL repository and invoke them dynamically, just as

[1] At the time of its definition, the acronym stood for Simple Object Access Protocol. In the standardized version, SOAP is no longer an acronym.

CORBA applications may discover an object's interface on the fly and invoke it using the Dynamic Invocation Interface. But the underlying development paradigm fundamentally differs. CORBA requires a developer to create an IDL description before implementing clients and servers that match the defined interface, whereas WSDL descriptions may be deployed for generating implementations but their use is not mandated. WSDL-compliant interface descriptions may be provided after the initial creation of the service. Additionally, the central storage of WSDL descriptions within a designated repository is not required by the Web Service paradigm. The service provider may also choose to serve WSDL descriptions at the physical location where the service is offered. When doing so, no standard repository has to be offered, although standard lightweight discovery mechanisms may be deployed [3].

WSA [5] defines core architectural concepts (e.g., discovery and life cycle) and relationships that are central to the interoperability of Web Services. It also defines a set of constraints and examines how the architecture meets the Web Services requirements expressed by stakeholders. Management issues are mentioned in the final document but the W3C has paid little attention to them. To date, this activity has not produced a formal standard prescribing a fixed architecture; it merely collected current concepts and terms and documented their relationships.

Choreography deals with the composition of Web Services and the description of relationships between Web Services. It is also known as orchestration, collaboration or coordination. The nascent W3C activity on choreography [12] is essentially based on Sun's Web Service Choreography Interface (WSCI) [1].

2.2 OASIS Activities

OASIS is an industrial consortium responsible for standardizing many domain-specific aspects of Web Services, especially ebXML, a markup language for e-business. The main general-purpose technology standardized by OASIS and relevant to integrated management is Universal Description, Discovery and Integration (UDDI). We will come back to UDDI later.

The OASIS Web Services Distributed Management Technical Committee recently began working on Web Services management. This activity covers two aspects: using Web Services to manage distributed resources and managing Web Services (the latter includes modeling a Web Service as a manageable resource).

In the area of choreography, the OASIS Web Services Business Process Execution Language Technical Committee took over the BPEL4WS [19] proposal by Microsoft, IBM *et al.* and is now standardizing it under the name Business Process Execution Language (BPEL). The main objectives of this work are to describe process interfaces for business protocols and define executable process models.

2.3 WS-I Activities

WS-I focuses on developing profiles, usage scenarios, use cases, sample applications and testing tools to facilitate interoperability between the Web Services platforms of its members. This industrial consortium began its activities in February 2002; few specifications have been released to date. The most significant is Basic Profile 1.0 [4], which consists of implementation guidelines as to how core Web Services specifications should be used together to develop an interoperable Web Services infrastructure. Management aspects have not been addressed yet by WS-I, but interoperability is critical for management systems.

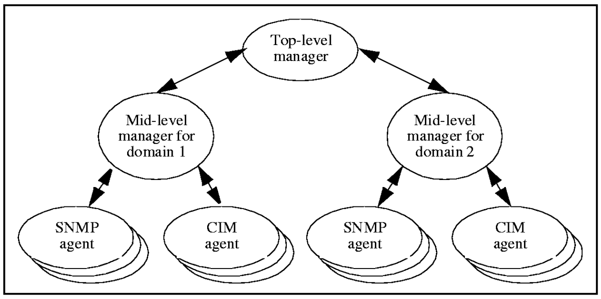

Top-level manager
Mid-level manager for domain 1
Mid-level manager for domain 2
SNMP agent
CIM agent
SNMP agent
CIM agent

Fig. 1. Distribution aspects in WIMA

3 WIMA and Design of JAMAP

Now that we have summarized the state of the art in Web Services, let us study the design of the open-source software used in this project: JAMAP. This research prototype of a management platform implements the Web-based Integrated Management Architecture (WIMA) [14] in Java. WIMA leverages XML's selfdescription capability to integrate SNMP[2] Management Information Base (MIB) data and Common Information Model (CIM) objects in a seamless manner, which is particularly useful in heterogeneous environments. WIMA is well suited to integrated management, that is, the integration of management data pertaining to network devices, systems, applications, end-to-end networks, services, etc.

In WIMA, agents publish the monitoring data and notifications they can send, and management applications (*managers*) subscribe to them in a semi- or fully automated way. The same publish-subscribe mechanism is used for manager-to-manager communication, when managers are organized hierarchically to manage a large domain or different domains. Data transfers are based on HTTP.

WIMA supports distributed management – to be precise, a weakly distributed hierarchical paradigm if we use the taxonomy defined in [14]. This allows administrators to split an organization into multiple management domains if the amount of management data to process grows large. Management domains may be defined according to many criteria: geographical location, management area (e.g., network management vs. service management), profit center, customer (e.g., when Internet Service Providers do virtual hosting), etc.

[2] SNMP is the Simple Network Management Protocol [18], currently the main standard for managing networks.

In the example depicted in Figure 1, the organization has adopted a three-tier management hierarchy. We have one top-level manager for the entire organization, one mid-level manager per management domain, and a number of agents (up to a few hundred) per domain.

In WIMA, the top-level manager runs the event correlator, which is the smart part of the management application. JAMAP 1.3 features a simple rule-based engine implemented in Java.

In WIMA, each mid-level manager is broken up into five components [14]:

– the *data analyzer*, which analyzes monitoring data on the fly, detects problems, and sends events to the event correlator when problems are detected;
– the *data collector*, in charge of collecting and filtering data on a regular basis for the purpose of monitoring; incoming data can be processed immediately by the data analyzer, archived in the data repository, or both;
– the *notification collector*, in charge of receiving incoming SNMP traps and CIM events, filtering them and forwarding them to the event correlator; it may also archive some of these incoming events in the data repository;
– the *configuration manager*, in charge of automatically configuring agents;
– the *background analyzer*, which performs data mining and non-realtime analysis on the data archived in the data repository; it may also send events to the event correlator when problems are detected.

WIMA allows each mid-level manager to be distributed across several physical machines. The mapping between the five previous components and physical machines is not constrained by WIMA.

In JAMAP 1.3, the first three components are implemented and the fourth is under development. Data collection, notification collection and data analysis are fully distributed, whereas event correlation is not. Each mid-level manager can comprise one or several data collectors (e.g., one for network management and another for service management), one or several notification collectors (e.g., one for SNMP traps and another for CIM events), and exactly one data analyzer.

Unlike most SNMP-based management platforms found to date on the market, JAMAP uses publish-subscribe (see Figure 2). SNMP MIBs supported by the agent are published in the data subscription applet, which can be downloaded by any management station. This applet communicates with the data subscription servlet inside the agent. This allows the manager to subscribe to specific MIB Object Identifiers (OIDs) and specify a push frequency for each OID. The data subscription servlet updates the push schedules in a persistent repository. From then on, management data is pushed regularly by the agent to the manager. The data is sent directly to the manager when we can run JAMAP software directly on the agent (e.g., when the agent is a Linux PC or a Windows PC); otherwise, data is pushed via a proxy (e.g., in our testbed, when the agent is a Cisco router).

When JAMAP was implemented, the focus was on the communication and organizational aspects. Release 1.3 includes no fancy GUIs and no dynamic discovery of the network topology. The event correlator supports a limited set of rule templates, and no state is currently retained by the rule engine between successive push cycles.

4 Why Use XML in Integrated Management?

The main reasons for using Web technologies in general, and XML in particular, in integrated management are analyzed in detail in [14] and only summarized here. Some of them are generic, others are specific to network and systems management.

4.1 Advantages of Using XML in General

Probably the main advantage of using XML in software engineering is that it is both standard and stable. It is backed by the W3C, which has a good track record of independence vis-à-vis any particular vendor's interests. Unlike Java, whose specifications changed several times a year for many years, XML has been through very few release cycles, and the ill-designed DTDs were swiftly replaced by XML schemas. Markup languages defined in XML come and go and may change often, but XML itself is very stable.

Second, it is platform-neutral and facilitates interoperability. The people who devised XML had learned from the mistakes of the Common Object Request Broker Architecture (CORBA); e.g., the Object Management Group (OMG), which is in charge of CORBA, took many years to standardize the Portable Object Adapter, even though important operations had not been standardized in the Basic Object Adapter and required platform-specific extensions.

Third, XML is simple and easy to learn. The learning curve of developers is usually short and training costs low (many XML tools are freely available).

Fourth, XML is widely used in industry. As a result, most software engineering students want to learn it, people who recently graduated usually know it, and many mature developers want to study it.

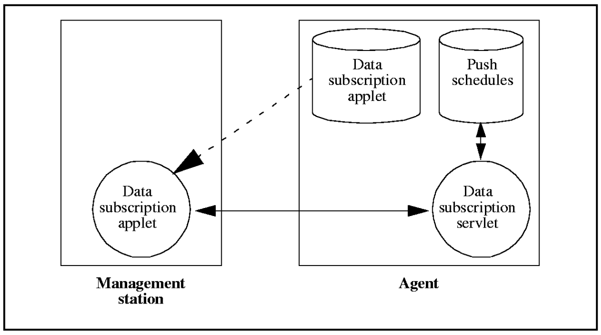


Fig. 2. Publish-subscribe

Fifth, using XML-based technologies is often a way to reduce costs. With XML and Web Services, organizations can often implement simple solutions based on loosely coupled middleware, instead of reengineering legacy applications and replacing legacy systems.

Sixth, W3C's XML Schema Definition (XSD) allows XML parsers to validate XML documents. This increases the robustness of XML-based distributed applications. Robustness has become a major concern in industry, as a growing number of enterprises entirely depend on the availability of their software systems to run their businesses.

Seventh, and more arguably, XML-based applications are easier to debug because XML is human readable. This argument is less compelling than it used to be, because a large number of XML documents now use metamodel mappings rather than model mappings [14].

4.2 Advantages of Using XML in Network and Systems Management

Middleware technologies have blossomed in the past decade. Until recently, when administrators purchased a management platform, they had to choose between CORBA, EJBs, .NET, proprietary middleware, etc. This variety of poorly compatible solutions increases development costs for vendors, who need to support multiple technologies; it augments purchase costs for customers, who pay for this variety even if they do not need it; but most of all, it decreases the safety and long-term visibility of customers' investments. If an administrator selects a middleware that is abandoned by his company a couple of years later, he will bear the responsibility for this "wrong" decision.

XML and Web Services allow for a "truce in the middleware war" [14]. Compared to full-blown object-oriented distributed environments, where everything must be an object, they keep a low profile. They can cope with object-oriented models at the edges but do not require them; they can also deal with data models; they can even extract data from old proprietary repositories designed several decades ago. By trying to achieve less and by successfully tackling interoperability from day one, XML-based technologies constitute a rather safe investment.

Another advantage of using XML in management is that it allows for a clean separation between information and communication models. The limitations of SNMP, which bundles the two, are explained in [14].

Third, as far as the communication model is concerned, XML is easy to use for representing management data in transit between agents and managers, or between managers in hierarchical management.

Fourth, regarding the information model, the success encountered by XML schemas is compelling: they have been adopted exceptionally quickly throughout the industry, and there is nothing specific to integrated management that should prevent this industry from leveraging XML. This message has been advocated by the DMTF for years; the IETF may soon be convinced, as evidenced by the recent work of the Network Configuration Working Group.

Fifth, as we experimented during this project, XML is appropriate for expressing persistent management data, especially configuration files, in a heterogeneous environment.

Sixth, XML facilitates the integration of multiple management areas (in our case, network management and systems management) by offering a general-purpose means of representing self-describing data, whatever the data. By doing so, it makes it easy to deal with the heterogeneity of information models found in real life.

4.3 Disadvantages of Using XML

XML is not the panacea, however. Its main disadvantages are threefold.

First, it is verbose. This increases network overhead, but also processing time and resource consumption at the edges. Although this is generally not an issue, as most layered software architectures used today are quite verbose, it can be problematic for resource-constrained equipment such as embedded systems, cell phones and inexpensive commodity devices. To cope with bandwidth problems arising from XML's verbose textual notations, W3C recently started the XML Binary Characterization Working Group, which deals with binary compression of XML content.

Another problem is that XML schemas and DTDs are so simple to create that vendors lack incentives to comply with standards. Why should self-describing data comply with XML schemas produced by slow-paced multi-vendor consortia, when they could be produced quicker and made publicly available by each vendor? Since it emerged in 1990, the SNMP management market has demonstrated that customers are not eager to use SNMP MIBs and information models produced by standards bodies: they want functionality. This problem can be alleviated by using Extensible Stylesheet Language Transformations [13], a generic and standard mechanism for transforming automatically instances of one XML vocabulary into another.

Last, validating an XML document takes time and consumes CPU and memory resources. Some agents cannot afford to do that. Therefore, management applications cannot always rely on validation. This is unfortunate because validating incoming XML documents is a proven way to make management applications more robust.

These disadvantages exist, but they are in our view outweighed by the advantages of using XML.

5 How to Use Web Services in a Management Platform

In the Internet world, most management platforms focus on four aspects [14]:

– *Regular management* consists of management tasks that run continuously, in pseudo-real time, over long periods of time. It encompasses monitoring, data collection (for background analysis), and notification handling.
– *Ad hoc management* consists of management tasks that run occasionally, if need be, for a short time. It comprises troubleshooting and short-term monitoring, and operates in pseudo-real time.
– *Configuration management* consists in changing the setup of an agent to make it operate differently.
– *Background analysis* includes all the management tasks that run in the background (as opposed to pseudo-real time) and strive to make sense of the data gathered by regular management (data collection). Examples include security analysis and the generation of daily usage reports.

For the sake of conciseness, let us focus on monitoring, a form of regular management that is implemented in JAMAP 1.3.

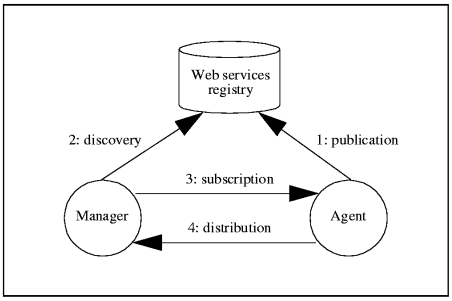


Fig. 3. Four phases of regular management

The four phases of monitoring (publication, discovery, subscription and delivery) are explained in detail in [14] and summarized in Figure 3. A priori, Web Services could be used at all levels in JAMAP:

- *Publication phase:* To allow agents to announce what data (e.g., SNMP OIDs or CIM objects) can be pushed to managers (respectively, in hierarchical management, to allow mid-level managers to announce what data can be pushed to top-level managers).
- *Discovery phase:* To allow managers to discover dynamically what management data can be pushed by agents (respectively, to allow the top-level manager to discover what data can be pushed by mid-level managers).
- *Subscription phase:* To allow managers to subscribe to data provided by agents by invoking Web Services directly on these agents (respectively, to allow the top-level manager to subscribe to data provided by mid-level managers).
- *Distribution phase:* To allow agents to push data to managers (respectively, to allow mid-level managers to push data to the top-level manager).

6 Publish-Subscribe and Discovery with UDDI

The UDDI technology is often advertised, or thought of, as a general-purpose technology implementing publish-subscribe for Web Services [20] [22]. As we needed such a mechanism in JAMAP, we studied, deployed, tested and evaluated UDDI.

To analyze the relevance of this technology for managing our testbed, we first adopt a top-down approach and study where UDDI registries allow us to store data useful for integrated management. Next, in a bottom-up approach, we model the management information that we want to store in a publish-subscribe registry and investigate how it fits with UDDI.

6.1 Top-Down Approach

Three versions of UDDI have been specified to date. UDDIv1 is now considered historic. When we conducted this work, the market was dominated by UDDIv2 and had not yet begun migrating to UDDIv3. The main new features in UDDIv3 are registry interaction and versioning [20].

After investigating a number of platforms listed in the Soapware directory [17] or mentioned in the `xml-dist-app@w3.org` mailing list, we selected Systinet's WASP UDDI [23], which offers a good balance between features, compliance with the latest specifications and free availability to researchers. The version that we tested (release 4.5.2) implements UDDIv2 and supports a few UDDIv3 features (e.g., subscriptions and notifications, which allow clients to automatically receive notification of changes made to registered Web Services).

The XML schema that underlies UDDI registries is rather simple. It consists of four elements:

– *businessEntity:* describes a business or an organization; the information provided here is equivalent to the yellow pages of a telephone directory: name, description, contact people, etc.
– *businessService:* provides a high-level description of a service provided by a company or an organization in business terms; this information is similar to the taxonomic entries found in the white pages of a telephone directory.
– *bindingTemplate:* provides a technical description of a given business service; it includes either the access point (e.g., a URL or an e-mail address) of this Web Service or an indirection mechanism that leads to the access point.
– *tModel:* technical models contain (i) pointers to technical documents used by developers of Web Services and (ii) metadata about these documents; they represent unique concepts or constructs, facilitate reuse and enable interoperability; they are primarily used as sources for determining compatibility between providers and consumers of Web Services and as keyed namespace references.

The first three elements are hierarchically structured: a business entity logically contains one or several business services, and a business service logically contains one or several binding templates. The fourth element lies outside this hierarchy: a binding template includes references to technical models (*tModels*) but does not contain the *tModels* themselves; as a result, a single *tModel* may be referenced by several binding templates.

The UDDI business entities and services are very coarse-grained. In practice, they can be used in two ways. First, a UDDI registry may be used to advertise the business services offered by a given business entity to other companies. A business publishes its core business activities and a potential customer may want to discover this information.

Alternatively, or sometimes concurrently, a UDDI registry may be used to announce within a company the services offered to its own staff. This use was not initially envisioned in UDDIv1 and UDDIv2, but the authors of UDDIv3 now advocate it as an important use of UDDI [20]. Corporations are sometimes organized into profit centers that run as independent businesses and charge each other. Using UDDI registries to discover what services are available within a corporation makes sense in such environments. For example, within CERN, IT and Administration (among others) offer services to all CERN staff. Both of them can therefore be modeled as business services within the business entity named “CERN".

Next in the UDDI hierarchy, the concept of binding template is more flexible than business entities and services: we can put more or less anything we want into it. So, binding templates are the only entities that we can define to make UDDI useful to manage network devices and systems.

6.2 Bottom-Up Approach

In JAMAP, publish-subscribe operates at a much finer-grained level of abstraction than UDDI business entities and services. For instance, a managed element may want to publish that it supports SNMPv2c and the version of MIB-II specified in RFC 1213; another may publish that it supports CIM Specification 2.2, CIM Core schema 2.7 and CIM System schema 2.7. How can we model that in a hierarchical way à la UDDI?

An intuitive information model would be as follows. Within the business entity “CERN", we find the business service “IT". Within the latter, we find the finer-grained service “DataTAG networking research". Within the latter, we find the service “gigabit network monitoring". Within the latter, we want to list all the agents that can be managed by JAMAP. Each agent then wants to announce the information models (SNMPv2c, CIM, etc.) that it supports. For a given agent (e.g., `w02gva.datatag.org`) and a given information model (e.g., “SNMP"), we want to advertise the list of data models (e.g., SNMP MIBs) supported by this agent. Since many agents only support portions of MIBs, we want agents to be able to publish what SNMP OIDs they support. And finally, we want to allow managers (programs) to subscribe to each OID at a given frequency (e.g., retrieve `ifInOctets` every 15 minutes).

This model has two shortcomings: the DataTAG project involves several research institutes, not just CERN, and we monitor a testbed network that does not belong to CERN.

A more suitable information model would be the following. Within business entity “European Union", we have another finer-grained business entity called “FP5/IST Projects". Within this entity, we find an element “project" of which “DataTAG Project" is an instance. Within a project, we find partners and activities. We model project partners by a sequence of XML elements each called partner; one of them is “CERN". So, our XML schema would already require four layers where UDDI only provides one: the business entity.

Next, project activities can similarly be modeled as a sequence of XML elements each called activity; one of them is “network monitoring". This activity can legitimately be modeled as a Web Service, since it is indeed a service offered to all project partners. Within this activity, we define two management domains: one called “CERN", which covers all the testbed systems and network devices located in Geneva; and another called “StarLight", which covers equipment in Chicago. In each management domain, we have agents, one per managed element. Each agent then advertises the information models that it supports. The rest of the information model is similar to the previous.

Unfortunately, this cannot be modeled using UDDI. We investigated whether finer-grained information could be stored in UDDIv2 and UDDIv3. We tried to leverage the *instanceParms* element of an *instanceDetails* structure in a *bindingTemplate*, to no avail.

With the simple information model currently supported by UDDI, we can only model that a company supports a management application and provides an access point for it at a given URL. This model is much higher level than what is required by JAMAP for the purpose of publish-subscribe and discovery between agents and managers. This conclusion is valid for all three versions of UDDI.

7 XML-Based Configuration Management in JAMAP

As UDDI registries are not appropriate to publish and discover agents in integrated management, we devised an XML schema for monitoring networks and systems with JAMAP, and used XML files to publish and discover management information.

7.1 Publication and Coarse-Grained Discovery

In JAMAP 1.3, a manager can discover all the agents within its domain by parsing an XML configuration file (`networkMap.xml`) that describes the organization's network. This file contains the addresses of the agents, management domain by management domain, and, for each agent, dynamic information such as the URL of its data collector servlet, an optional proxy address, and the URL of the agent's configuration file (`agentManagement.xml`).

To increase robustness, we strived to be precise in the XML schema definition file (`networkMap.xsd`). For instance, an IP address is not simply a string: we give precise definitions of IPv4 and IPv6 addresses. This allows JAMAP to validate effectively the XML documents exchanged between managers and agents. In JAMAP 1.3, publication is still manual.

7.2 Fine-Grained Discovery

Each agent has an `agentManagement.xml` file associated with it. This file may be published by the agent itself or another machine. It can be validated against the tailor-made XML schema mentioned above.

The `agentManagement.xml` file contains agent-specific information that a manager needs to know in JAMAP (see Figure 4). First, we find information pertaining to the configuration Web Service of the agent (access point and WSDL file location). Next, we find the URL of the Java applet used for manual configuration. Then, we define the URLs of a number of Java servlets run by the agent: one for configuring the agent, another for pushing data to the manager, etc. Last, we specify the information models and data models that are supported by the agent. For the SNMP information model, each entry indicates the RFC defining a specific version of an SNMP MIB, the community string, and the encoding: Basic Encoding Rules, XML, serialized Java, plain text, etc. For the CIM information model, each entry includes the name of the schema, its version number, and the encoding.

One advantage of our XML schema is that it makes it easy to add new information models or agent-specific information. Another is that it allows for strong validation of XML documents.

```
<agentManagement>
  <accessPoint>
    http://137.138.35.18:8080/services/AgentConfigurationService
  </accessPoint>
  <WsdlFileLocation></WsdlFileLocation>
  <dataSubscriptionApplet>
    http://137.138.35.18:8080/DataSubscriptionApplet.jsp
  </dataSubscriptionApplet>
  <subscriptionSheetServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.SubscriptionSheetServlet?
  </subscriptionSheetServlet>
  <getServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.Get?
  </getServlet>
  <pushDispatcherServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.PushDispatcherServlet?
  </pushDispatcherServlet>
  <agentConfigurationServlet>
    http://137.138.35.18:8080/servlet/jamap.servlet.AgentConfigurationServlet?
  </agentConfigurationServlet>
  <informationModels>
    <snmpInformationModel>
      <rfc>/mibs/rfc1213-mib.txt</rfc>
      <snmpv1CommunityString>public</snmpv1CommunityString>
      <encodingType>plainText</encodingType>
    </snmpInformationModel>
  </informationModels>
</agentManagement>
```

Fig. 4. Example of `agentManagement.xml file`

7.3 Subscription

Subscriptions can be either manual or automated. If a subscription is done manually, we can call directly the subscription Web Service on the agent by using the information retrieved from the previous files. Alternatively, we can use an applet if a URL is specified. When subscriptions are automated, we just need to edit an XML subscription file that is located at a certain URL and contains, for each agent, the subscribed data and its push frequency.

The automation of subscriptions relies on simple criteria, as illustrated by the following examples:

- in management domain “CERN", for all devices of type “Linux PC‘" supporting the SNMP information model, retrieve the `ifInOctets` and `ifOutOctets` columnar objects every 15 minutes if the PC supports RFC 1213 (MIB-II), and retrieve the `hrSWRunPerfCPU` and `hrSWRunPerfMem` columnar objects every 5 minutes if the PC supports RFC 1514 (Host Resources MIB);
- in all management domains, for all devices of type “Cisco 76xx", retrieve the `ifInOctets` and `ifOutOctets` columnar objects every 5 minutes.

8 Lessons Learned

A number of lessons of general interest can be learned from this case study and may hopefully prove useful to other projects.

Easy to use: As claimed by many Web enthusiasts, XML is indeed easy to use and debug. The Simple API for XML (SAX) makes it very easy to parse an XML document in Java and validate it against an XML schema. Tomcat [21], the Apache open-source package that implements Java servlets, is simple to use and well documented.

Web Services: Axis [2], the Apache incarnation of Web Services, works in simple cases but still suffers from teething problems. Web Services discovered in a WSDL repository cannot be invoked dynamically if they use complex types (other than string and integer); instead, one has to use stubs à la CORBA. This explains why `WsdlFile-Location` is empty in Figure 4. Also, invoking a Web Service from within an applet requires the Java applet security scheme to be turned off, unless one uses commercial security certificates. Hopefully, future versions of Axis will address these issues.

Portability: XML is highly portable and is very appropriate for defining configuration files used by management applications. Similarly, both Tomcat and Axis are portable: they work fine on Linux 2.4.20, Windows XP and Windows 2000 platforms.

SOAP: SOAP can be used to push data between managers and agents but it offers little flexibility. A SOAP toolkit usually comes as a black box: there is often no easy way to control the HTTP connection underneath (e.g., for setting socket or TCP options, or for using long-lived HTTP connections). This is a problem for JAMAP, since we want to avoid the overhead of frequently setting up and tearing down HTTP connections.

Discovery: There are currently no standard ways of publishing, discovering and subscribing to fine-grained Web Services for integrated management. UDDI is too coarse-grained for our purposes. Hopefully, a new solution will come up in the future.

In this paper, we have described how we implemented Web Services in JAMAP and summarized the lessons learned in this process. Our conclusions are fourfold: Web Services are suitable for managing network devices and systems; XML portability facilitates integration in a heterogeneous environment; UDDI is a white-page service for e-commerce and is too coarse-grained for managing network devices and systems; and we still lack a standard way of publishing, discovering and subscribing to monitoring services between managers and agents.

In the future, it would be useful to devise a generic mechanism to publish, discover and subscribe to management Web Services. This mechanism should make it possible for management application designers to use any XML schema, as opposed to a fixed schema as in UDDI. Another interesting challenge would be to allow generic components of a management application to discover and bind to one another by using Web Services. Would the flexibility offered by component software be outweighed by the decrease in performance?

Acknowledgments. This research was carried out while P.A. Doffoel was an M.Sc. student at ENSIMAG doing an internship at CERN. Part of this work was funded by the FP5/IST Program of the European Union (DataTAG project, grant IST-2001-32459).

References

1. A. Arkin, S. Askary, S. Fordin et al., (Eds.), *Web Service Choreography Interface (WSCI) 1.0*, W3C Note, World Wide Web Consortium, 2002. Available at `http://www.w3.org/ TR/2002/NOTE-wsci-20020808/`.
2. Axis, `http://ws.apache.org/axis/`.
3. K. Ballinger, P. Brittenham, A. Malhotra *et al.*, (Eds.), *Web Services Inspection Language (WS-Inspection) 1.0*, IBM, 2001. Available at `http://www-106.ibm.com/ developerworks/webservices/library/ws-wsilspec.html`.

4. K. Ballinger, D. Ehnebuske, M. Gudgin et al., (Eds.), *Basic Profile Version 1.0*, Final Material, Web Services Interoperability Organization, 2004. Available at http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html.
5. D. Booth, H. Haas, F. McCabe et al., (Eds.), *Web Services Architecture*, W3C Working Group Note, World Wide Web Consortium, 2004. Available at http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.
6. R. Chinnici, M. Gudgin, J.J. Moreau et al. (Eds.), *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, W3C Working Draft, World Wide Web Consortium, 2004. Available at http://www.w3.org/TR/2004/WD-wsdl20-20040326/
7. DataTAG Project, http://www.datatag.org/.
8. M. Gudgin, M. Hadley, J.J. Moreau *et al.* (Eds.), *SOAP 1.2 Part 1: Messaging Framework*, W3C Candidate Recommendation, World Wide Web Consortium, 2002. Available at http://www.w3.org/TR/soap12-part1/.
9. M. Gudgin, M. Hadley, J.J. Moreau et al. (Eds.), *SOAP 1.2 Part 2: Adjuncts*, W3C Candidate Recommendation, World Wide Web Consortium, 2002. Available at http://www.w3.org/TR/soap12-part2/.
10. M. Gudgin, A. Lewis and J. Schlimmer (Eds.), *Web Services Description Language (WSDL) Version 2.0 Part 2: Message Exchange Patterns*, W3C Working Draft, World Wide Web Consortium, 2004. Available at http://www.w3.org/TR/2004/WD-wsdl20-patterns-20040326/.
11. JAMAP 1.3, http://www.datatag.org/jamap/.
12. N. Kavantzas, D. Burdett and G. Ritzinger (Eds.), *Web Services Choreography Description Language Version 1.0*, W3C Working Draft, World Wide Web Consortium, 2004. Available at http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.
13. M. Kay (Ed.), *XSL Transformations (XSLT) Version 2.0*, W3C Working Draft, 2003. Available at http://www.w3.org/TR/2003/WD-xslt20-20031112/.
14. J.P. Martin-Flatin, *Web-Based Management of IP Networks and Systems*, Wiley, 2002.
15. Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/.
16. J. Schönwälder, A. Pras and J.P. Martin-Flatin, "On the Future of Internet Management Technologies", *IEEE Communications Magazine*, Vol. 41, No. 10, pp. 90–97, Oct. 2003.
17. Soapware.org, http://www.soapware.org/.
18. W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*, 3rd Edition, Addison-Wesley, 1999.
19. S. Thatte (Ed.), *Business Process Execution Language for Web Services (BPEL4WS) Version 1.1*, 2003. Available at http://www.ibm.com/developerworks/library/ws-bpel/.
20. The Stencil Group, *The Evolution of UDDI—UDDI.org White Paper*, July 2002.
21. Tomcat, http://jakarta.apache.org/tomcat/.
22. UDDI.org, *UDDI Technical White Paper*, Sept. 2000.
23. WASP UDDI, http://www.systinet.com/products/wasp_uddi/overview/.
24. Web Services Interoperability Organization (WS-I), http://www.ws-i.org/.
25. World Wide Web Consortium (W3C), http://www.w3.org/.

A Conceptual Comparison of WSMO and OWL-S

Rubén Lara[1], Dumitru Roman[1], Axel Polleres[1], and Dieter Fensel[2]

[1] Digital Enterprise Research Institute (DERI) Innsbruck
Innsbruck, Austria
{ruben.lara, dumitru.roman, axel.polleres}@deri.org
http://www.deri.at/

[2] Digital Enterprise Research Institute (DERI) International
dieter.fensel@deri.org
http://www.deri.org/

Abstract. Web Services have added a new level of functionality on top of current Web, enabling the use and combination of distributed functional components within and across company boundaries. The addition of semantic information to describe Web Services, in order to enable the automatic location, combination and use of distributed functionalities, is nowadays one of the most relevant research topics due to its potential to achieve dynamic, scalable and cost-effective Enterprise Application Integration and eCommerce. In this context, two major initiatives aim to realize Semantic Web Services by providing appropriate description means that enable the effective exploitation of semantic annotations, namely: WSMO and OWL-S. In this paper, we conduct a conceptual comparison that identifies the overlaps and differences of both initiatives in order to evaluate their applicability in a real setting and their potential to become widely accepted standards.

Keywords: Web Services, Semantic Web, Ontologies, Semantic Web Services, Mediators.

1 Introduction

This paper presents a conceptual comparison between the Web Service Modeling Ontology (WSMO) [8] and the Semantic Web Services ontology OWL-S [6]. WSMO, which is based on WSMF [1], and OWL-S, are the most salient initiatives to describe Semantic Web Services, aiming at describing the various aspects of Semantic Web Services in order to enable the automation of Web Service discovery, composition, interoperation and invocation.
The comparison presented in this paper exposes the conceptual relation between the latest stable versions of WSMO and OWL-S, i.e. WSMO-Standard v0.1 [8] and OWL-S v1.0 [6]. A comparison between DAML-S, the predecessor of OWL-S, and WSMF [1] (the framework which served as a starting point for WSMO) is presented in [2].

The purpose of this comparison is to highlight the conceptual overlaps and conceptual differences between the two specifications. Please notice that the conducted comparison is done at a conceptual level. It will guide a formal mapping between the two specifications when more mature versions of WSMO are available and its missing elements defined in more detail. Section 2 presents a more detailed motivation of the comparison. Section 3 discusses the purpose and principles of each initiative. Using OWL-S as the reference point, Section 4 conducts a comparison between the elements described in OWL-S and the ones considered in WSMO. The elements introduced in WSMO and not present in OWL-S are described in Section 5. A summary of the relation between WSMO and OWL-S is presented in Section 6. Finally, Section 7 summarizes our conclusions and future work.

2 Motivation

Web Services have added a new level of functionality to the current Web, making the first step to achieve seamless integration of distributed components. Nevertheless, current Web Service technologies only describe the syntactical aspects of a Web Service and, therefore, only provide a set of rigid services that cannot adapt to a changing environment without human intervention. The human programmer has to be kept in the loop and scalability as well as economy of Web Services are limited [1].

The vision of Semantic Web Services is to describe the various aspects of a Web Service using explicit, machine-understandable semantics, enabling the automatic location, combination and use of Web Services. The work in the area of Semantic Web is being applied to Web Services in order to keep the intervention of the human user to the minimum. Semantic markup can be exploited to automate the tasks of discovering services, executing them, composing them and enable seamless interoperation between them [5], thus providing what are also called intelligent web services.

The description of Web Services in a machine-understandable fashion is expected to have a great impact in areas of e-Commerce and Enterprise Application Integration., as it can enable dynamic, scalable and reusable cooperation between different systems and organizations. These great potential benefits have led to the establishment of an important research area, both in industry and academia, to realize Semantic Web Services.

Two major initiatives have to be considered in this context: The first one is OWL-S [6], an effort by BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, SRI International and Yale University to define an ontology for semantic markup of Web Services. OWL-S, currently at version 1.0, is intended to enable automation of web service discovery, invocation, composition, interoperation and execution monitoring by providing appropriate semantic descriptions of services. The second recent initiative is the Web Service Modeling Ontology - (WSMO–Standard, currently at version 0.1) [8], an initiative to create an ontology for describing various aspects related to Semantic Web Services aiming at solving the integration problem.

WSMO is led by the Semantic Web Services working group of the SDK cluster[1], which includes more than 50 academic and industrial partners.

Both initiatives have the goal of providing a world-wide standard for the semantic description of Web Services, grounding its application in a real world setting. Due to the wide intended audience of these initiatives, it is essential to identify their overlaps and differences in order to evaluate their real potential to realize the vision of Semantic Web Services and to be adopted in real applications.

In the following sections, we provide a comparison that identifies the similarities and differences of WSMO-Standard and OWL-S that can be used to evaluate which initiative provides better means for the semantic description of Web Services with respect to different aspects.

3 Purpose and Principles of WSMO and OWL-S

Before comparing in detail the description elements introduced by WSMO and OWL-S, we present in this section the purpose and principles of both initiatives, as this provides an important view of the problems WSMO and OWL-S intend to solve and how they conceptually ground the proposed solution.

3.1 Purpose

The purpose of OWL-S is to define a set of basic classes and properties for declaring and describing services i.e. an ontology for describing Web Services that enable users and software agents to automatically discover, invoke, compose and monitor Web resources offering services, under specified constraints [6]. OWL-S tries to cover the description of services in a wide sense, not focusing on a particular application domain or problem. WSMO aims to create an ontology for describing various aspects related to Semantic Web Services, but with a more defined focus: solving the integration problem. WSMO also takes into account specific application domains (e-Commerce and e-Work) in order to ensure the applicability of the ontology for these areas.

3.2 Principles

The development of OWL-S is centered on a set of tasks to be fulfilled, namely: automatic Web Service discovery, invocation, composition, interoperation, and execution monitoring. Although OWL-S inherits the foundations of research areas such as planning and agents, it does not explicitly delimitate what principles are applied to the development of the ontology. WSMO is more explicit in this regard, as it is grounded on a number of explicit principles of the domain that the ontology formally represents. Furthermore, WSMO is based on the conceptual work done in WSMF [1], and aims to follow from this conceptualization all the way to a formal execution

[1] http://sdk.semanticweb.org/

model backed up by an open source implementation[2]. In general, WSMO presents a conceptually more focused approach than OWL-S, making its specific underlying conceptual work and principles more explicit.

4 Comparison OWL-S / WSMO

In this section, we analyze each of the description elements introduced in OWL-S and identify the equivalent or similar concepts in WSMO (if any). OWL-S defines an upper ontology for the semantic description of Web Services. The core elements of this upper ontology are depicted in Figure 1.

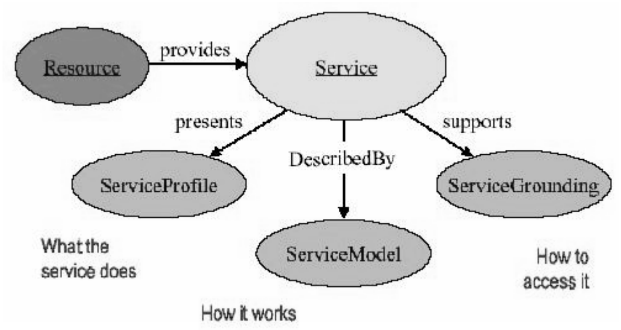


Fig. 1. OWL-S upper ontology

The Service concept serves as an organizational point of reference for declaring Web Services, and every service is declared by creating an instance of the Service concept. WSMO does not provide such an organizational reference, particularly, goals are viewed separately from Web Services and the core concepts of the ontology - goals, services, and ontologies - can be defined separately (although they may refer to other elements previously defined using special services called mediators).

In this sense the elements defined in WSMO-Standard are loosely coupled, in contrast with the stronger coupling of OWL-S. OWL-S also defines a Resource concept that defines the resource that provides the service and the resources used by it. WSMO-Standard does not define such a concept. OWL-S Resource overlaps the implementation and business layers, as the resource providing the service and the resources used by the service should be at different levels. In contrast, the next development step of WSMO - WSMO-Full - will clearly separate the implementation layer (WSMO-grounding) and the business layer on top.

In the remainder of this section, we go through the core elements in OWL-S and analyze their correlation with the elements of WSMO.

[2] Initial efforts for the development of a reference implementation for WSMO are already in progress, see http://www.wsmx.org

4.1 OWL-S Service Profile

In OWL-S, the Service Profile describes the intended purpose of the service, both describing the service offered by the provider and the services needed by the requester. This concept is split into two different views in WSMO: the goal and the Web Service capability. A goal in WSMO specifies the objectives that a client may have when she/he consults a Web Service. A Web Service capability defines the service by means of what the service offers to the requester i.e. what functionality the service provides. Therefore, WSMO distinguishes the provider point of view and the requester point of view, providing different elements to express the service capability and the requester needs. The OWL-S service profile can be expressed by the combination of the WSMO goal, the WSMO Web Service capability, and the Web Service non-functional properties. In the following list, we go through the details of the OWL-S service profile and present their counterparts in WSMO.

Profile cardinality. OWL-S links a Web Service to its profile via the property *presents*. The ontology does not dictate any minimum or maximum cardinality for this property, so the service can present no profile or several profiles. WSMO allows linking Web Service to none or several goals by using different mediators, and to have an empty capability for the service. WSMO does not allow a service to present more than one capability, but the link of a capability to several goals by using different mediators can be seen as the definition of multiple service profiles in OWL-S.
Registry. OWL-S does not dictate any form of registry for services. This also applies to WSMO, where no explicit registry architecture for any of its elements is defined. For both formalisms the currently most likely option seems to be extensions of or embedding in UDDI, such as suggested in [3,7].
Service name, contact and description. The OWL-S service profile includes human-readable information, contained in the properties *serviceName*, *textDescription* and *contactInformation*. This information is expressed in WSMO by using non-functional properties (core properties), such as *title, description, identifier, creator or publisher,* that can be defined for the goal and for the Web Service capability.
Functionality description. The OWL-S profile specifies what functionality the service provides, the specification of the conditions that must be satisfied for a successful result, and the results of the service execution. This is described by using four different elements: inputs, outputs, preconditions and effects.
Information transformation is expressed in OWL-S using inputs and outputs. This is described in WSMO by using pre-conditions and post-conditions. The goal only includes post-conditions, defined as the state of the information space that is desired. The capability defines both pre-conditions (what the service expects for enabling it to provide its service, defining conditions over the input) and post-conditions (what the service returns in response to its input, defining the relation between the input and output).
State change is expressed in OWL-S using preconditions and effects. This is described in WSMO by using assumptions and effects. The goal only defines effects, as the state of the world that is desired. The capability defines both assumptions (similar

to pre-conditions, but referencing aspects of the state of the world beyond the actual input) and effects (the state of the world after the execution of the service).
OWL-S' conditional outputs and effects can be equally expressed in WSMO, as post-conditions and effects are defined using a logical expression. It can be seen that the functional aspects of the OWL-S profile can be described in WSMO. In addition, other functional aspects not covered in OWL-S can be expressed. This will be discussed in Section 5.
Profile attributes. The OWL-S profile contains attributes that specify non-functional properties of the service. These include an expandable list of non-functional properties and a service category (using a categorization external to OWL-S). In contrast, WSMO does not offer an explicit list of expandable non-functional properties. Rather it defines a list of core non-functional properties and some extensions e.g. for the Web Service, based on the Dublin Core Metadata Element Set. While not mentioning an explicit mechanism for the expansion of these lists, these core non-functional properties can be extended freely by subclassing to reflect extra non-functional properties of any of the modeling elements used. An external service categorization is also not explicitly considered in WSMO, since such a categorization can again be achieved naturally by introducing a respective taxonomy subclassing of the respective WSMO concepts. In this sense, WSMO elements can be extended in any direction, which makes WSMO less explicit in this regard but more flexible.

4.2 OWL-S Service Model

In OWL-S, a Service Model represents how the service works. The service is viewed as a process, and the class ProcessModel is the root class for the definition of the service process. The process model represents functional properties of the service, together with details of the composition of the service (if the service is a composite service). In WSMO, the Web Service concept plays the role of the process model, specifying the details of the functionality of the service together with other aspects such as non-functional properties or service interfaces. In the following, we go through the details of the OWL-S service model and present their counterparts in WSMO.
Model cardinality. OWL-S links a Web Service to its model by using the property describedBy. A Web Service can be described by at most one service model and it does not impose any minimum cardinality. In WSMO, a Web Service can have 0 or 1 capability, and 0 or multiple interfaces. As the interface in WSMO defines the orchestration of the service, the same service can operate in different ways.
Functionality description. In OWL-S, a Web Service is viewed both as a data transformation process i.e. a transformation from a set of inputs to a set of outputs, and as a state transition in the world i.e. some effects in the world emerge after the execution of the service[3].

[3] OWL-S definitions are misleading in this regard, as both aspects imply a state change, in the information space and in the state of the world space, respectively.

Information transformation. OWL-S does not restrict the number of inputs or outputs of a service, and conditions for the delivery of outputs can be specified. In WSMO, the Web Service capability also captures the information transformation performed by the service, by means of (a not limited number of) pre-conditions and post-conditions. OWL-S conditional outputs can be captured as part of the WSMO post-conditions. WSMO pre-conditions and post-conditions are wider than OWL-S inputs and outputs. They can be seen as a superset of those, as they add more details on the information transformation performed by the Web Service. This will be discussed in Section 5.
State change. OWL-S does not restrict the number of preconditions or effects of a service, and conditions under which these effects emerge can be specified. In WSMO, the Web Service capability also captures the world state change caused by the execution of the service, by means of (an arbitrary number of) assumptions and effects. OWL-S conditional effects can be captured as part of the WSMO effects definition.
Profile/Model consistency. It can be seen above that the functionality description of the Web Services is specified both in the service profile and in the service model. The consistency between the service profile and the service model is not imposed in OWL-S. The descriptions contained in the profile and in the model can be inconsistent without affecting the validity of the OWL expression (although it may result in a failure to provide the service functionality). This is also the case in WSMO, where the link between the service capability and the service goal i.e. the wgMediator does not contain any requirement on the consistency of the post-conditions and effects defined in the goal and the ones defined in the service capability. However, the OWL-S specification envisions that the set of inputs, outputs, preconditions and effects (IOPEs) of the service profile are a subset of the ones defined by the service model, while WSMO does not make any assumption on this regard. This increases WSMO flexibility in the definition of requester needs and service capabilities and provides a looser coupling of these elements because mediators can serve to add missing conditions.
Atomic processes. OWL-S atomic processes can be invoked, have no sub processes, and are executed in a single step from the requester's point of view. They are a subclass of the process model, so they specify their inputs, outputs, pre-conditions and effects. In WSMO, an atomic process can be defined by the capability of the service, or it can be a link to an external Web Service (defined as a proxy in the orchestration of the Web Service).
Simple processes. OWL-S simple processes are not invocable and they are viewed as executed in a single step. They are used as elements of abstraction. The concept of simple process is not explicitly present in WSMO, although it could be represented as a proxy that links to a Web Service with no grounding.
Composite processes. They represent processes that are decomposable into other processes. Such composition is specified using several control constructs defined in OWL-S. WSMO can model OWL-S composite processes by defining the service orchestration and proxies for the constituent sub-processes as part of the service interface. Nevertheless, WSMO does not provide at its current status any means to define the orchestration of the services i.e. it does not provide any control flow to define a composite service using proxies. The same applies for the definition of the data flow.

OWL-S provides a process annotation to define the data flow between different processes, whereas this is still under defined in WSMO. Thus, in future versions of WSMO it is intended to have a rich notion of state and state transition based on an Abstract State Machines-like notation.. Particularly, it is planned to extend the current description towards an explicit separation between the modeling of orchestration and choreographies of the service, where the latter shall describe the externally observable behavior of the service. This is currently covered by the messageExchange primitive in WSMO which shall allow specifying patterns beyond the basic input-output primitives currently available in OWL-S, whereas the orchestration describes the internal behavior of the services, which does not necessarily have to be published. To some extent, choreographies can be modeled by OWL-S process models, but an explicit abstraction from the internal composition only exposing the external process behavior is not foreseen and also each service can only expose at most one process model, while WSMO shall offer the possibility to declare different external behaviors via multiple interfaces. However, these plans are still to be realized and not part of the early version of WSMO that serves as the basis for this comparison.

4.3 OWL-S Service Grounding

The grounding in OWL-S gives the details of how to access the service, mapping from an abstract to a concrete specification of the service. In WSMO, a Web Service also includes a grounding that defines the access to the service.
Grounding cardinality. OWL-S links a Web Service to its grounding by using the property supports. A Web Service can have multiple groundings (although an atomic process can have only one grounding) and a grounding must be associated with exactly one service. These groundings are associated to the atomic processes defined in the service model, although this association is not described in the model but only in the grounding. Therefore, the groundings for the atomic processes of the model can only be located by navigating from the service model to the service (via the describes property), and from there to the service grounding (via the supports property). In WSMO, the groundings are linked to the Web Service via the groundings property, and a Web Service can have multiple groundings. How these groundings relate to the orchestration of the service is not specified at the current version of WSMO.
Grounding definition. OWL-S does not dictate the grounding mechanism to be used. Nevertheless, the current version of OWL-S only provides a pre-defined grounding for WSDL, mapping the different elements of the Web Service to a WSDL interface. WSMO follows the same approach and does not mandate any grounding mechanism. However, WSMO does not provide a clear specification for the grounding mechanism, while OWL-S already provides an explicit mapping to WSDL at its current version.

5 Additional Elements in WSMO

In this section, we present the additional elements that are present in WSMO but not in OWL-S.

5.1 WSMO Non-functional Properties – Core Properties

As indicated above, WSMO introduces a set of core non-functional properties that are defined globally and that can be used by all the modeling elements of WSMO. These properties include the Dublin Core Metadata Element Set[4] plus a version element. OWL-S does not define this kind of globally accessible and pre-defined non-functional properties, but, as related in Section 4.1, it defines an expandable list of non-functional properties in addition to some non-functional properties such as service name and contact, in the profile, but they are not used for other modeling elements.

5.2 WSMO Ontologies

In WSMO ontologies are described at a meta-level. A meta-ontology allows describing all the elements of the ontologies used for the modeling of services. An ontology in WSMO consists of non-functional properties (from the set of pre-defined core properties), used mediators (that enable modularization by dealing with the ontology import problems, such as alignment, merging or transformation of ontologies), concept definitions, relation definitions, axioms, and instances. For a detailed description of the meta-ontology we refer to [8].

OWL-S does not define such a meta-ontology to ground the definition and use of other ontologies in addition to the Web Service description concepts. In addition, OWL-S does not provide any common conceptualization of the domain ontologies required to describe the service, not dealing with the problems that may arise when importing different ontologies. By importing other OWL Ontologies in OWL (and OWL-S being an OWL ontology itself), there is the possibility to reuse existing vocabulary in OWL-S descriptions. However, describing ontologies on a meta-level such as it is done in WSMO offers more flexibility not dictating the formalism (i.e. the specific ontology language) to be used in describing imported ontologies. Moreover, we conceive the mediator concept underlying WSMO more general and powerful than the OWL import primitive to combine/include existing ontologies.

5.3 WSMO Goals

Goals are defined in WSMO as the objectives that a client may have when he consults a Web Service. Goals consist of non-functional properties (from the set of pre-defined core properties), used mediators, post-conditions and effects. OWL-S profiles have

[4] http://dublincore.org/documents/dces/

the role of expressing the requester needs, but also the role of expressing the service capabilities. They do not provide a separate conceptualization for requester needs and service capabilities, which makes the OWL-S approach less flexible.

Goals in WSMO contain neither pre-conditions nor assumptions, capturing only the results (both in terms of information and state change) that the requester requires. This separation can be done in OWL-S by expressing a partial profile i.e. only with outputs and post-conditions. Some non-functional properties defined in the goal are not considered in the OWL-S profile.

WSMO, through the use of mediators in the goal definition, allows the importing of external ontologies for the definition of the goal in an explicit and controlled manner, while OWL-S does not provide this feature for profiles. Furthermore, goals in WSMO can be defined by reusing existing ones, allowing the refinement of those or the combination of several pre-existing goals. This is achieved by using goal-goal mediators. OWL-S does not provide facilities for the reuse of profiles beyond subclassing which makes the possibilities for combination of profiles to define new ones rather limited.

5.4 WSMO Mediators

As one of its pillars, WSMO introduces the concept of mediators in order to bypass heterogeneity problems. For an introduction to the concept of Mediators, cf. [11]. Mediators in WSMO are special services used to link heterogeneous components involved in the modeling of a Web Service. They define the necessary mappings, transformations or reductions between the linked elements. As depicted in Figure 2, four different types of mediators are defined, namely:

ggMediators. They link two goals, expressing the (possible) reduction of one of the goals, and the mediators employed to import the terminology (ontologies) used to define the goal. WSMO allows linking not only goals, but also goals to ggMediators, thus allowing the reuse of multiple goals to define a new one. This facility is not present in OWL-S, where the definition of profiles by using existing ones is limited to the OWL subclassing mechanism.

ooMediators. They import ontologies and resolve possible representation mismatches among all imported ontologies. As explained before, this explicit import mechanism that deals with the problems that may arise when importing ontologies, is not present in OWL-S.

wgMediators. They link a Web Service to a goal. They can state the differences or the reductions between the two elements and they can map different vocabularies. In OWL-S, this possibility does not exist, as the Web Service is linked to one or more profiles via the property presents, which does not contain any information about functionality reduction or heterogeneity problems. It can be seen that OWL-S implicitly assumes that a profile is defined uniquely for each service, while WSMO defines goals independently and through the use of mediators enable the reuse of requester goals, and the link of a service to different goals, explicitly considering possible reductions and the inherent heterogeneity of distributed environments.

wwMediators. They link two Web Services, containing the ooMediators necessary to overcome the heterogeneity problems that may arise when the services use different vocabularies. Again, OWL-S does not consider this kind of mediation, and different Web Services are linked through the control constructs of the Service Model, including no mediation information.
In general, WSMO explicitly contains modeling elements to bypass heterogeneity problems that necessarily arise in the domain it tries to cover, while OWL-S leaves this issue unresolved and does not provide any explicit way to overcome the heterogeneity in the description of Web Services.

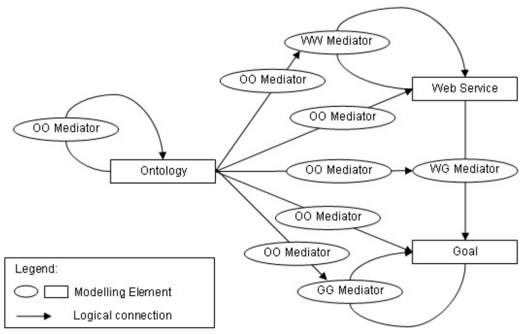


Fig. 2. Mediators in WSMO

5.5 WSMO Web Services

Web Services in WSMO are defined in addition to elements such as ontologies, goals and mediators, while in OWL-S the Service concept is the organizational point of reference. Therefore, WSMO provides a loose coupling of its modeling elements as opposed to the tighter coupling in OWL-S: The description elements in OWL-S are centered on the concept of service, which in turn results in a tighter coupling of these elements. For example, it is implicitly enforced that profiles are uniquely defined for each service, as no mediation mechanism is defined between the service model and the service profile. In addition to this organizational aspect, the definition of Web Services in WSMO contains some elements that are not found in OWL-S.
Non-functional properties. WSMO defines non-functional properties for Web Services in addition to the core properties presented before. They include aspects related to the quality of the service, such as performance, reliability, security or trust. OWL-S includes all the non-functional properties of the service in the profile, mainly as an expandable list of non-functional properties. In this sense, OWL-S is more explicit in its extension mechanism, but as discussed in Section 4.1 this extendibility can be achieved equally in WSMO (by subclassing).
Used mediators. The definition of the Web Service includes the ooMediators used by the service to import its required terminology. As explained in Section 5.4, OWL-S does not provide any explicit mechanism for importing this terminology and dealing with the problems that may arise when importing ontologies.

Capability. The capability of a service is defined separately from goals. It includes the ooMediators required to import the capability terminology, and wgMediators that links the capability to a goal, dealing with the possible reduction of functionality, reduction of non-functional properties, and heterogeneous terminologies. The capability of the service defines the functional aspects of the service offered, modeled by using pre-conditions, assumptions, post-conditions and effects. While WSMO assumptions resemble OWL-S pre-conditions and WSMO effects are very close to OWL-S effects, some differences arise in the other modeling elements. Moreover WSMO foresees arbitrary axioms in pre-conditions, post-conditions, assumptions and effects whereas OWL-S limits conditions to OWL class expressions at its current state. Extensions towards a fully fledged rule language like SWRL[5] or DRS[6] are under discussion.

Pre-conditions. Pre-conditions in WSMO are more encompassing than OWL-S inputs, as they include conditions over the input, providing a more detailed description of what the Web Service expects to be able to provide its service.

Post-conditions. Post-conditions in WSMO are more encompassing than OWL-S outputs, as they include the relations between the input and the output. Such relations cannot directly be modeled in OWL-S, making the OWL-S definition of the service functionality less expressive than formalizing arbitrary axioms relating input and output as in WSMO. In the last version of OWL-S some simple means of relating inputs and outputs and conditions are provided by so called parameter bindings but as long as OWL-S does not provide means for arbitrary logical expressions or rules, functionality description remains limited.

Interfaces. The interfaces of services give details about the operation of the service. They introduce aspects not considered in OWL-S.

Errors. Errors of different types can arise when using a service. WSMO explicitly models the error information of the Web Service, while OWL-S does not consider these details. Although errors can be captured by using conditional outputs, this characterization of errors is not explicit, as the definition of a conditional output does not necessarily imply that one of the possible outputs is an error.

Orchestration. The orchestration can be defined in OWL-S by using the service model. However, OWL-S does not allow the use of profiles to define characteristics of the service needed while performing its actual location and invocation at run-time. WSMO does introduce the concept of proxy in the orchestration definition. As proxies can be a goal or a wwMediator, it means that both dynamic and static compositions are possible in WSMO, while OWL-S dynamic composition is limited. On the contrary, as discussed above, means of defining orchestration of the service is still underspecified in WSMO, and more elaborated in OWL-S.

Compensation. WSMO includes wwMediators or goals to define the compensating services in case of error. However, OWL-S does not consider compensation in its description of a Web Service.

[5] http://www.daml.org/rules/proposal/

[6] http://www.daml.org/services/owl-s/1.0/DRSguide.pdf

Message Exchange. Exchange Message exchange patterns are defined in WSMO to describe the temporal and causal relationships, if any, of multiple messages exchanged. OWL-S does define data and control flow, which can be used to model these kinds of patterns. OWL-S does not enable the reuse of pre-defined message exchange patterns. We have elaborated on this point earlier in Section 4.2
Grounding. WSMO allows defining several groundings for the same Web Service, while OWL-S imposes that an atomic process must have exactly one grounding, which limits the number real implementations an OWL-S Web Service can have. This leads to some problems when defining real services, as described in [10].

5.6 WSMO Layering

Due to the complexity and difficulty of the domains Semantic Web Services are applied to, WSMO follows a layered approach. There are three WSMO species envisioned: WSMO-Lite, WSMO-Standard and WSMO-Full. WSMO-Lite represents a minimal yet meaningful subset (in the context of web service integration) of WSMO-Standard for which an execution environment is easily implementable, whereas WSMO-Full aims to extend WSMO-Standard and to incorporate a B2B perspective.
OWL-S does not follow such layered approach in the modeling of Services but defines a single ontology for describing Web Services. As OWL-S is based on OWL, and OWL has also three species (OWL-Lite, OWL-DL and OWL-Full), OWL-S inherits this layering from OWL. However, this layering is given by the language used and not by the complexity or intended application of the Web Service ontology. So, this layering is orthogonal to the layering in WSMO.

5.7 WSMO Language

F-logic [4] is suggested to be used in WSMO to express the logical expressions defined in goals, mediators, ontologies and Web Services. However, for the definition of the ontology itself, WSMO does not dictate any representation language. On the contrary, while OWL-S is based in OWL from the beginning and as the representation language for ontologies, there is no agreement yet on how to specify logical expressions such as conditions, etc. Currently F-Logic syntax is used to declare the ontology defined by WSMO, but it could equally be expressed using OWL; no restriction has been made so far regarding the ontology language used. The use of F-Logic to express a logical expression, which is the range of some properties in WSMO, does not impose any formalism for the definition of the underlying ontology.

6 Summary

The results of our comparison show that most of the description elements defined in OWL-S can be modeled in WSMO. Some aspects, especially the specification of the service orchestration and the WSDL grounding, are more detailed in OWL-S. However, it is foreseen that successive versions of WSMO will further specify these under-defined aspects. It can also be seen that WSMO covers aspects e.g. mediation that are not considered in OWL-S and that increase the applicability of WSMO in a real setting. Table 1 provides a summary of the core comparison elements. Notice that it does not include every element compared in this paper but the most relevant results.

Table 1. Core results of the comparison

Comparison aspect	WSMO-Standard	OWL-S
Purpose	Focused goal, specific application domains	Wide goal, does not focus on concrete application domains
Principles	Explicit conceptual work and well-established principles	Not explicit, development based set of tasks to be solved and foundations inherited from other research areas
Coupling	Loose coupling, independent definition of description elements	Tighter coupling in several aspects
Extensibility	Extensible in every direction	Limited extensibility, mainly through OWL subclassing
Implementation & business layers	Will be clearly separated in WSMO-Full	Overlapped at some points e.g. use of the Resource concept
Registry	Not dictated	Not dictated
Requester needs & service capabilities	Two different points of view, modeled independently and linked via wgMediators	Not separated, unified view in the service profile
Functionality description	Explicit and complete description	Does not describe some aspects of the functionality
Non-functional properties	Pre-defined properties. Flexible extension but not explicit mechanism	Few pre-defined properties. Explicit extension mechanism but improvable flexibility
Orchestration	Supports static and dynamic composition, but under-defined	Limited dynamic composition, completely defined
Grounding	Multiple groundings, not pre-defined grounding	Problems with multiple groundings for atomic processes, WSDL pre-defined grounding
Mediation	Scalable mediation between loosely coupled elements	No mediation
Layering	3-layers (WSMO-Lite, WSMO-Standard, WSMO-Full) covering different complexity levels of the domain	No layering (layering inherited from OWL, does not reflect complexity of the application domain)
Languages	F-Logic for logical expressions. Ontology language not imposed	Language for conditions not defined. Ontology language OWL.

7 Conclusions and Future Work

WSMO covers most of the description elements introduced in OWL-S, and introduces additional elements that increase its applicability in real domains. Aspects such as mediation and compensation are key issues to be solved for the realization of Semantic Web Services, and to make this technology applicable for e-Commerce and e-Work.

Additionally, during this comparison we identified yet underdefined details of web service description formalisms to be added. In particular, future versions of WSMO should provide a higher level of detail for the definition of aspects such as choreography or grounding. If these elements are appropriately covered, WSMO can become a strict superset of OWL-S that also covers relevant issues not covered by OWL-S. WSMO also intends to have an execution platform, called WSMX, while the intentions of OWL-S in this direction are not yet defined.

At their respective current versions, OWL-S and WSMO show a different level of specification. OWL-S is already at version 1.0, whereas the version of WSMO underlying this comparison is only 0.1, leaving certain details less defined than OWL-S. However, the more explicit and detailed conceptual grounding of WSMO and its design rationale seems to be promising for the development of a standard for the modeling of Semantic Web Services.

Our future work will concentrate on following the evolution and further development of both OWL-S and WSMO. We will update this comparison accordingly. A relevant update to this comparison will be the inclusion of the use case introduced by the WSMO working group, as a key comparison element to determine the applicability of both initiatives to a real domain, and to provide a formal mapping between WSMO and OWL-S.

Acknowledgements. The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, COG and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The authors would like to thank to all the members of the WSMO working group for their advice and input to this work.

References

1. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications. 1(2). 2002.
2. Flett, A.: A comparison of DAML-S and WSMF. Technical report. 2002.
3. Herzog, R., Lausen, H., Roman, D., Zugmann, P. (eds.): WSMO Registry. WSMO working draft, available at http://www.wsmo.org/2004/d10/v0.1/, 2004
4. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object oriented and frame-based languages. Journal of the ACM. 42(4):741-843. 1995.

5. McIlraith, S., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems. 16(2). March/April. 2001.
6. The OWL-S Service Coallition: OWL-S: Semantic Markup for Web Services, version 0.1. Available at http://www.daml.org/services/owl-s/1.0/owl-s.pdf.
7. Paolucci, M., Kawamura, T., Payne, T. R., Sycara, K.: Importing the Semantic Web in UDDI. Proceedings of Web Services, E-business and Semantic Web Workshop, 2002.
8. Roman, D., Keller, U., Lausen, H.(eds.): Web Service Modeling Ontology (WSMO), available at http://www.wsmo.org/2004/d2/v01/index.html.
9. Roman, D., Lausen, H., Oren, E., Lara, R. (eds.): Web Service Modeling Ontology – Lite (WSMO-Lite), available at http://www.wsmo.org/2004/d11/v01/index.html.
10. Sabou, M., Richards, D., Splunter, S.: An experience report on using DAML-S. WWW 2003 workshop on E-services and the Semantic Web (ESSW'03). Budapest, Hungary. 2003.
11. Wiederhold, G.:Mediators in the Architecture of Future Information Systems. IEEE Computer 25(3):38-49 , 1992.

A Conceptual Framework for Semantic Web Services Development and Deployment

Claus Pahl

Dublin City University, School of Computing
Dublin 9, Ireland
Claus.Pahl@dcu.ie

Abstract. Several extensions of the Web Services Framework have been proposed. The combination with Semantic Web technologies introduces a notion of semantics, which can enhance scalability through automation of service development and deployment. Ontology technology – the core of the Semantic Web – can be the central building block of this endeavour. We present a conceptual framework for ontology-based Web service development and deployment. We show how ontologies can integrate models, languages, infrastructure, and activities within this framework to support reuse and composition of semantic Web services.

Keywords: Web Services, Semantic Web, ontology technology, conceptual development and deployment framework.

1 Introduction

Opening the Web for software applications is the objective of the Web Services Framework WSF [1]. Services are self-contained computational entities, used as is by service requesters, and made available through the infrastructure provided by a provider. The focus is on the boundaries of systems and on the interaction between systems.

A number of shortcomings of the Web Services Framework WSF can be identified [2,3,4,1]. On the structural level, composing services is not part of the WSF. The description of services is limited to syntactical and type aspects; no support is provided for functional and non-functional semantical properties. The combination with the Semantic Web [5], in particular ontology technology [6,7], can provide an essential step forward that introduces meaning to Web services and that provides the foundations to enable a software component-style composition of services [8,9].

Previous work on the combination of the Semantic Web and Web Services has often focussed on modelling and language aspects [2,3]. More architecture-oriented treatments have neglected the semantical aspects [1]. Here, our aim is to identify the central aspects of a conceptual framework for semantic Web services architectures [10]. We address models, languages, infrastructure, and stakeholder activities – and illustrate the integrating role that ontology technology can play in this endeavour. Such a framework can form the underlying foundation of a methodology for semantic Web services development and deployment. It provides a taxonomy for a Web services development and deployment platform. A major aim of the framework is to link models, languages, infrastructure, and activities. The conceptual framework results from an analysis of our own language-oriented work [11,12], but also related work on semantic Web services and Web services infrastructures such as [13,14,2,15,16,17,3,18,19,20,4, 21,22,23,10]. We aim to capture these in a generic conceptual framework.

In our framework, particular models for services are essential and need to be prescribed. This is a clear deviation from the WSF focus on interfaces and interactions. Our notion of a Web services architecture is connected to a different style of service development and deployment than anticipated by the WSF – based on principles such as composition and reuse and a notion of services as processes.

Automation of stakeholder activities in a shared and distributed environment, such as the discovery and selection of suitable services for a requester, requires a new distributed type of development and deployment methodology in the Web services environment based on joint activities, sharing knowledge and artefacts, and reuse – supported by a distributed architecture geared towards this purpose.

In Section 2, we outline the basics of Web services and introduce the rationale behind our conceptual framework. In Section 3, we investigate model and language aspects of a conceptual framework for Web services. Infrastructure and activity aspects of the framework are subject of Section 4. We end with some conclusions.

2 Web Services

Our objective is to introduce a conceptual Web services framework supporting semantic service reuse and composition. We propose ontology technology as a means to integrate successful techniques used in WSF extensions – domain modelling [3] or design-by-contract [2] – into a coherent framework.

2.1 A Conceptual Framework for Service-Oriented Architectures

The Web Services Architecture WSA [1] defines ”a Web service [as] a software system identified by a URI, whose public interfaces are defined and described using XML. Other systems may interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.” A wider scope of the service notion includes distributed object – or Web-mediated – services, i.e. software agents providing the service functionality that are not necessarily part of the Web environment themselves[1].

Services in service-oriented architectures (SOA) are coherent collections of operations described in an interface and provided to a user. Often a service is

[1] This would allow us to see the WSF as a meta- or interoperability framework between middleware platforms.

seen as an abstract notion that must be implemented by a concrete agent [1]. There are consequently two aspects of services:

- The *internal view*. Services provide functionality through operations. These operations might encapsulate an internal state; their behaviour certainly needs to be coherent in terms of the service they provide.
- The *external view*. Two different roles – those of requesters and providers – are immediately apparent. The interaction between these – either humans or software agents acting on their behalf – is a central aspect. For instance, agreement on the service semantics and the mechanisms of message exchange are vital.

Both the internal and external view need to be looked at in the context of service development and deployment. A *conceptual framework* provides an abstract model of the development and deployment context that integrates the various aspects, including underlying conceptual models, languages, development and deployment infrastructure, and activities of the stakeholders involved:

- *Models and Languages*. These provide the foundations necessary to model services as coherent sets of operations. All service aspects relevant for a potential user need to be captured in abstract descriptions. In public environments, representing and sharing knowledge is central.
- *Infrastructure and Activities*. Specific interactions are required between requester and provider – activities such as discovery of services, composition, and invocation of service agents. These have to be supported by an adequate infrastructure consisting of protocols and tools.

The *service development and deployment aspects* (model, language, infrastructure, and activity) form the different *layers* of our *conceptual framework* – see Fig. 1.

Service-oriented architectures (SOA) are based on remote procedure calls RPC, adding a publication and discovery infrastructure. Our framework suggests an extension of SOAs towards a service-oriented development and deployment architecture by adding further development infrastructure, e.g. semantics and composition.

2.2 Semantic Web Services

Different development scenarios for services involving requester and provider can be imagined: collaborative development of services or provider-based development of services with human or automated discovery. In any case, the existence of requester and provider makes sharing of knowledge about services and their context necessary.

Often, the automation of development and deployment processes involving Web services – from the discovery to the final invocation – is seen as the ultimate goal [3]. The degree of automation in this context determines the scalability of the framework. A cornerstone of such an endeavour is the support of semantics

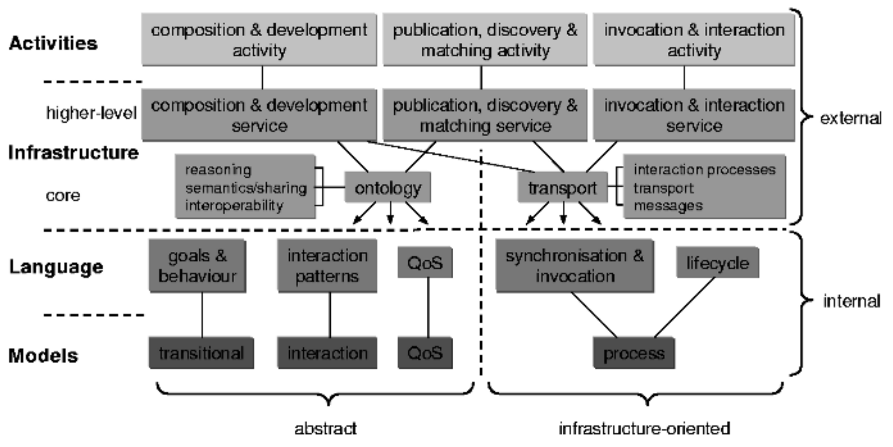


Fig. 1. A Layered Conceptual Framework for Semantic Web Services

for services [3,2,16,19,4,22,18,23,21,14,20,13,15]. The WSF focuses on message format and message exhange mechanisms to provide and invoke services. In addition, various semantical properties of services are relevant for a service user, e.g. [24,2,11]:

– *Transitions*: the abstract behaviour is often represented in a transitional form describing in/out-transitions.
– *Dependencies*: the interaction of a requestor with a service might be constrained, i.e. operations can only be invoked following an *interaction pattern* or *protocol*.
– *Interaction processes*: the internal process and interaction structure, possibly involving other services, that provide the functionality for the service. While similar to the external interaction patterns, this needs to address *data and control flow* and *synchronisation* more explicitly.

Semantical properties – including behaviour and dependencies – enable the reuse of services and their independent composition, resulting in a different development and deployment style for Web services. We will ignore quality-of-service attributes [21] – which can range from software attributes such as maintainability, security, and efficiency to aspects such as pricing.

2.3 Extending the Web Services Framework

In the WSF [1], a description language, WSDL (Web Services Description Language), is used to describe syntactical and type aspects of services, in particular message formats and message exchange details, and their binding to a communications protocol. A registry service, UDDI (Universal Description, Discovery, and Integration), allows providers to publish service descriptions and service requesters to search for services. A protocol, SOAP (Simple Object Access Protocol), is used to invoke services.

However, there is no support for semantical descriptions or the composition of services, and (business) processes cannot be modelled. Some attempts have already been made to rectify this [25,2,14,13,15]. For instance, BPEL4WS [14] and WSFL [25] are Web service flow languages that allows Web services to be composed (choreography and orchestration); DAML-S (now also known as OWL-S) [2] is a Semantic Web-compliant ontology for Web service description, and the Web Service Modelling Framework WSMF [3] is an ontology-based modelling approach.

Taking the concepts of these approaches on board, we can identify essential aspects of Web services that should form core elements of a Web services descriptions:

– *external descriptions* of the service in terms of its *goal* or *purpose* (assumptions and characterisation of the expected outcomes in terms of domain concepts), the *effect* (how acceptable input is transformed into output), and interaction protocols (ordering of operations) – these aspects often form the *contractual information*,
– *internal descriptions* including data and control flow that coordinates interactions between subservices,
– *interaction infrastructure descriptions* for services consisting of input/output data formats and ports and the protocol binding to handle the message exchange.

We can distinguish profile information (what a service requires and provides), model information (how the service works), and binding information (how the service is used).

We propose to follow the route taken by OWL-S and WSMF towards an extended WSF and base our framework on ontology technology. Moreover, we add a new aspect [8,9,26]. Component technology aims at modular composition of software systems from self-contained, reusable components described by contract-based interfaces and explicit context dependencies. Looking at component technology explains our motivation. WSDL descriptions do not make dependencies on other services explicit; they do not state their infrastructure requirements – which would, however, be a prerequisite for reuse and independent composition.

The Web services architecture WSA [1] focuses on messages, i.e. sees the description of message formats and their exchange at the core of the architecture, rather than the effects that are caused by message exchange. We focus on service semantics in the context of service choreography and orchestration where dependencies have to be made explicit. The automation of activities such as discovery is a central aim in both cases[2]. Whereas the aim of the WSA is not

[2] We use the term 'activities' here instead of 'processes', which we will use later on in a more technical sense

to prescribe particular development approaches, we will introduce here specific models and techniques for construction, discovery, and choreography of services – a consequence of chosing a specific (ontology-based) framework for service semantics.

We will discuss the central framework aspects – model and language, infrastructure and activities – in the next two sections. The discussion will always refer to Fig. 1. We refer to the literature to indicate the origin of and rationale behind framework elements.

3 Models and Languages

The semantic description of a service in a shared knowledge representation format based on common domain and computation models is a central element of a conceptual services framework. Knowledge engineering becomes therefore a pivotal technology.

3.1 Ontology Technology

The Semantic Web initiative aims at making the Web more meaningful and open to manipulation by software applications [5]. A logic-based approach based on knowledge representation and reasoning forms the backbone. Annotations to Web resources express meaning, which can be used by software agents to extract semantical information about the resource. The requirement for this to work is a precise, shared understanding of these annotations.

Ontologies provide a solution for this requirement. Ontologies define terminologies and semantical properties. Essentially, ontologies are hierarchical definitions of concepts of a domain and descriptions of the properties of these concepts. Logics such as description logics [7] provide the reasoning support. Integrated into a ontological Web framework based on OWL – the Web Ontology Language – sharing of ontologies becomes possible [6].

Two types of knowledge relevant to the services context need to be represented ontologically. *Domain knowledge* captures entities from the application domain and their properties – domain modelling is a widely accepted requirements engineering method. *Software knowledge* captures software artefacts and their properties. Expressing semantics and reasoning about it is the central goal. Software knowledge is often expressed by incorporating domain knowledge. Description and reasoning facilities provided by ontologies are essential building blocks of our conceptual framework.

3.2 Service Models

In particular computational aspects of service properties need to be based on appropriate models that underlie semantical description and reasoning.

In previous sections we have outlined the types of information needed to adequately represent service behaviour, including input/output behaviour, interaction protocols, and service composition and communication. Three types of computational models can address these aspects [14,20]:

– *Transitional model.* An abstract view on services and in particular on service operations is the transitional input-output behaviour. These descriptions are often called contractual information; pre- and postcondition-based techniques are usually used [27].

  A suitable model that covers the contractual aspects of the service is a state-transition model defining operations as transitions in a state space.
– *Interaction model.* An abstract view on a service's interactions with a service user. Often, only certain interaction patterns based on the offered operations are possible. Constructors such as sequence, choice, and iteration can be used to formulate these interaction protocols.

  Again, a state-transition model, here with constructors to compose transitions, is suitable to model the interaction behaviour of a single service or operation and to capture the service interaction patterns.
– *Process model.* A more detailed view on interactions between services is needed, viewing services as interacting processes. The interaction between a service and its user and also between the internal subservices used to provide the overall service needs to be addressed [12]. In both cases, the focus is on sending and receiving messages, and on the synchronisation between processes.

  A classical process model, as formulated in process algebras, can form the basis here to cover process synchronisation aspects for service invocations.

Quality-of-Service models complement the range of models [21]. Due to their variety, we neglect a detailled description here.

We can classify the conceptual models (and the corresponding languages) into two categories: *abstract* and *infrastructure-based.* Only the process model falls into the latter category. The service model acts as a conceptual model, outlining essential modelling requirements arising from the Web services context, but also as a semantical model in which descriptions can be interpreted. The modelling requirements expressed through these models have to be reflected in ontology languages. For some of these aspects, extensions of a classical ontology language might be required.

3.3 Abstract Service Description

We suggest an ontology language – at the core of a variety of semantic Web services approaches such as [3,2] – to introduce a description notation for abstract Web service properties. We use a description logic here, which underlies a language such as OWL [28]. Description logics are based on the idea of defining a concept in terms of its properties in relation to other concepts.

Describing Services as Processes. Central to the modelling facet of the framework is to understand services as processes [14,20]. Service processes and service-oriented composition have been identified as weaknesses in the current WSF. A process view allows us to include process interaction. Moreover, it helps us to formalise (and eventually automate) stakeholder activities. Consequently, ontologies describing service properties in their domain context need to focus on processes as the central entities.

Description logic [7] knows two basic elements. *Concepts* are classes of objects with the same properties. For instance, in an banking application, an *Account* is a central concept. *Roles* are relations between concepts. Roles express properties of concepts in relation to other concepts. An *Account* can be characterised by a *balance*-property, which relates *Account* with a *Numerical* value concept.

Concept descriptions are constraints based on simple set-theoretic operators and quantified expressions. *Operators* include $\top, \bot, \neg, \sqcup, \sqcap$ and $\rightarrow$ with their usual set-theoretic meaning. For instance, *PrivateCustomer* $\sqcup$ *CommercialCustomer* is the concept that describes the union of both customer classes. The *value restriction* $\forall R.C$ for a given concept C' restricts the values of role R (as a relation) to elements that satisfy C; the *existential quantification* $\exists R.C$ requires the existence of a role value satisfying C. For instance, an *Account* could be characterised by $\exists balance.Numerical$.

Different ways to model services have been suggested. In [2,16,22], services are represented as concepts, with properties associated through roles. In [11, 12], services are modelled as roles, interpreted by accessibility relations between states. Essential is to provide operators to compose services based on the idea of services as interacting processes.

Goals and Behaviour. We can associate pre-state and post-state descriptions with services. Properties of these states (in different formats) can be expressed using roles.

- *Goals* are abstract specifications about service behaviour [3]. *Assumptions* are pre-state properties that summarise basic concepts definitions relevant to the service, such as account or balance. The goal itself is an expression of the expected outcome of a service execution, usually involving the assumed concepts. An example is the expectation that after lodging money into an account, the balance will have increased.
- *Contractual information* about behaviour can be specified in terms of pre- and postconditions [2]. These conditions are expressions relating to parameters of the service operation signature, possibly involving domain concepts. For a lodgment service, the sum transfered into an account plus the pre-state balance yields the post-state balance. Contracts are refinements of goals [3].

Often, extensions of a classical ontology language [7] are necessary to enable goal and in particular contractual specifications. In [11,12], it is necessary to introduce names into role expressions in order to express parameters. An example is

$$\forall.lodgment \circ \underline{Sum}_N; postCond.equal(Bal, pre\text{-}Bal{+}Sum)$$

saying that a transitional role *lodgment* is applied to parameter name *Sum*, and that after execution the balance *Bal* is increased by *Sum* (which is the postcondition).

Interaction Protocols. Interaction protocols are pattern expressions constraining the order in which operations of a service can be invoked. In order to facilitate these expressions, we need introduce the operators, e.g. ; (sequence), + (choice), ! (iteration) to support the *interaction model* [12]. For instance,

$$open; !(lodge + transfer)$$

expresses that after opening an account, money can be repeatedly lodged or transfered.

3.4 Infrastructure-Based Service Descriptions

Service Synchronisation and Invocation. In order to deal with synchronisation and actual interactions described in the *process model*, we need to take another view on service operations [14,20]. So far only considered as transitions in a state-based systems, we need to consider both the requester and the provider of these transitions [29,30]. For instance, an automation of accesses to UDDI repositories would require such a process communication view. A description notation will build up on the ontology language for abstract service description by adding process calculi elements.

- *Ports.* The operation names define ports that, if synchronised with another port from another process, can form an interaction channel.
- *Orientation.* Each port carries additional information indicating whether it is used for sending or receiving on the channel. We use $op(a)$ for input (receiving) and $\overline{op}\langle a\rangle$ for output (sending) following [29] instead of an abstract role expression such as $op \circ a$ that we have introduced in Section 3.3.

The expression

$$getBalance(bal); \overline{newBalance}\langle bal + ldg\rangle$$

based on the abstract expression

$$getBalance \circ \underline{bal}_N; newBalance \circ (\underline{bal}_N + \underline{ldg}_N)$$

expresses that the specified service receives input from *getBalance* that it uses internally and then returns the result $bal + ldg$ to the service client using port *newBalance*.

Service Lifecycle. A notation to express service process interaction can be used to formalise an *activity-based lifecycle view on services* – which leads us into the infrastructure and activity aspects of our conceptual framework. Addressing the complete software lifecycle is an essential aspect of software engineering methodologies [31]. A service lifecycle is determined by activities such as matching, composition, and execution, and supported by infrastructure facilities such as repositories, brokers, and protocols. The lifecycle can be expressed as a process where different ports represent the infrastructure to support activities. A service port actually facilitates several activities:

– *Contract.* Using contract ports, matching constraints guard the establishment of an invocation infrastructure using the different type of invocation ports.
– *Invocation* and *Reply.* Invocation ports allow a service to be invoked and necessary parameters to be passed. Message type aspects constrain this interaction. Often, a service reply is communicated on another channel.

A provider lifecycle based on these service port types could follow the pattern

$$\textsc{Pro}\ s_C(s_I); !(\ \textsc{Exe}\ s_I(a, s_R); \textsc{Rep}\ \overline{s_R}\langle f(a)\rangle\ )$$

with annotations for providing Pro, executing Exe, and replying Rep for the contract, invocation, and reply ports s_C, s_I, and s_R, respectively [11]. The interaction pattern expresses that, after a contract match, a service can be invoked and a reply can occur an arbitrary number of times. This would formalise the UDDI-supported matching of WSDL descriptions of Web services and their invocation using SOAP in the WSF [1].

4 Infrastructure and Activities

The core task of a SOA infrastructure is to facilitate service invocation, but it also needs to support other stakeholder activities. The basic requirements for our conceptual framework arises already from the architecture required for discovery and invocation in the WSF. Semantic description and composition services can be layered on top.

4.1 Infrastructure Services and Facilities

Infrastructure services and facilities can be divided into core and higher-level:

– *Core infrastructure services* are *transport* – the distribution technology – i.e. the layered infrastructure model, and *ontologies* – the knowledge and semantics technology – i.e. the layered ontology model.
– *Higher-level services* build up on core services. *Publication, discovery, and matching* – based on transport and ontologies service – support discovery based on semantics-enabled UDDI and WSDL. *Development and composition* – based on transport and ontologies service – support composition and choreography based on semantics and interaction. *Service invocation and interaction* – based on the tranport service – support interaction for service invocation based on e.g. SOAP.

These services are usually provided through suitable APIs. Tools such as repositories, brokers, composition engines, and protocols facilitate these services.

We will discuss the higher-level services supporting development and deployment activities now in more detail. Distribution is the a property of a Web services architecture. Both development and deployment activities take place in this distributed context.

The central activities of invocation and execution of Web services shall be based on a *layered infrastructure model* for *transport* [32]. Starting at the bottom, message types characterise the payload of *messages*. *Transport bindings*, e.g. SOAP, define the message layout. Exchange-related aspects – protocol properties such as resending rules – are covered. The essential aspect are the *interaction processes* – defining the sequencing of send and receive operations.

4.2 Development and Deployment Activities

A Web services architecture needs to enable stakeholders (providers and requesters) to carry out development and deployment activities. Building up on core infrastructure services (transport and ontologies) activities such as discovery, composition, and invocation and interaction need to be facilitated. A simplified process based on discovery, matching and invocation/interaction activities can be modelled through a lifecycle protocol; see Section 3.4.

Publication, Discovery, and Matching. Requesters need to find and compare service providers for the services they need. The infrastructure that the WSF provides is the UDDI registry. Providers can publish descriptions of their services in these registries which can then be searched.

The central difficulty is *matching* [33], i.e. to find the service(s) that most closely match the requirements of the requester. In an automated setting, a software agent will use requirements formulated by the requester in a shared ontology language to search repositories for matching services.

A notion of matching needs to capture the idea of satisfaction or refinement [34]. A provided service needs to be at least as good as the requested one. In an ontology language, a *subsumption* concept – the subclass relationship between concepts or roles – captures this. A service matching notion needs to be composite, as services themselves and also their descriptions are composite [22]. For each of the individual aspects we need some kind of metric to decide matching. Each of them is supported by an underlying conceptual model (Section 3.2).

– *Goals and contractual information* – based on a transitional model. For instance, refinement-based notions of matching can be used; weakening the precondition and strengthening the postcondition is a standard choice [35, 33,34].
– *Interaction protocols* – based on an interaction model. A notion of simulation can form the basis of matching [30].
– *Processes* – based on a process model. A notion of simulation can again form the basis of matching here.

In all cases, the matching constructs can imply subsumption and can therefore be integrated into an ontological framework, see [11,12].

Subsumption allows us to capture widely used software development concepts such as specialisation and refinement. An enhancement could be achieved if another crucial relationship, the part-whole relationship, is addressed. For example, the meronymy concept falls into this category. It addresses parts standing for the whole – something that happens if a service operation is refered to, but the whole service is actually meant.

Description and reasoning using ontology technology is the central contributor to discovery and matching activities. Reasoning can be facilitated through the use of description logic-based inference tools [7] or through the use of transition system and automata-based approaches for verification [36].

Composition and Development. Composition can be both a development-time and a run-time activity. Services can be composed to composite services. We can distinguish client-side and provider-side composition of provider services, and provider-side constraining of provider services followed by client-side composition [8,9]. The variants can be characterised by the degree of cooperation and the degree of automation that is enabled. Automation is important for run-time composition.

Invocation and Interaction. Internet protocols provide the basic infrastructure. On top of these, service-specific protocols such as SOAP provide an RPC mechanism. The ontology-based interaction patterns describe the interaction behaviour of services.

In an automated approach, activities of the provider and the requester have to be synchronised. We can define inference rules based on the ontology language that govern these synchronisations at runtime. Important is here that different communication channels are used for retrieval and matching on the one hand and later service invocation interactions on the other – as expressed in the service lifecycle example.

5 Conclusions

Semantic Web services are now an increasingly important topic. Several directions – including domain modelling and composition – are currently investigated. However, two central problems remain. Firstly, a coherent, integrating conceptual framework is lacking. In particular, how to integrate the different aspects models, language, infrastructure and activities, is not adequately addressed. Secondly, services as processes is a notion that is central to enhance the Web services framework – and that needs to be made explicit in conceptual frameworks and service architectures supporting these frameworks. Service choreography and orchestration are two terms that capture the idea of business and workflow process definition based on service process composition.

A high degree of automation is a requirement for the future of the Web services framework – scalability and, therefore, the success of the framework depends on it. Automation requires shared semantics in a distributed, hetergeneous environment for development and deployment. Ontology technology is a solution to these problems[3]. Ontologies are reflected in all facets of our conceptual framework – models, languages, infrastructure, and activities. Ontologies can capture the process-oriented view on services and provide the necessary features to support the corresponding activities.

The proposed conceptual framework is the result of an investigation into various approaches in the context, guided by our own work. It aims to act as a taxonomy and through its linkage of models, languages, infrastructure, and activities, it helps us to better understand the problems of Web services development and deployment. It captures current developments such as the Web service framework WSF, OWL-S, BPEL4WS, and others and places these in the wider development context. The framework promotes the idea of a lifecycle-oriented approach to Web services development.

Our analysis of a number of semantic Web services extensions indicates progress towards a new methodology for service development and deployment – to be supported by a generic conceptual and architectural framework. The Web environment requires suitable workflow processes in particular for service development. Our proposed services-oriented development and deployment framework is different in many ways from the Web services framework WSF. Firstly, it exhibits characteristics of a component framework. Secondly, it supports a different style of development and deployment embracing composition and workflow processes. It creates a space for composable, Web service-enabled components. Our achievement is the introduction of a framework for these service components that connects the facets model, language, infrastructure, and activity based by coherent Web-based ontology and transport technologies.

One of the aspects that we neglected in our discussion are quality-of-service issues. They include various aspects including performance and security. Security in particular is of paramount importance. In [37], we have already explored the extension of the WSF by security requirements descriptions. However, the integration with infrastructure technologies requires more attention.

References

1. World Wide Web Consortium. *Web Services Framework.* http://www.w3.org/2002/ws, 2003.
2. DAML-S Coalition. DAML-S: Web Services Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.
3. D. Fensel and C. Bussler. The Web Services Modeling Framework. Technical report, Vrije Universiteit Amsterdam, 2002.
4. J. Peer. Bringing Together Semantic Web and Web Services. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.

[3] The technical aspects of the ontology framework we have presented here are only indicative of what is needed on the language and model side.

5. W3C Semantic Web Activity. Semantic Web Activity Statement, 2002. http://www.w3.org/sw.
6. H. Kim. Predicting How Ontologies for the Semantic Web Will Evolve. *Communications of the ACM*, 45(2):48–54, Feb 2002.
7. F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.
8. C. Szyperski. *Component Software: Beyond Object-Oriented Programming – 2nd Ed.* Addison-Wesley, 2002.
9. I. Crnkovic and M. Larsson, editors. *Building Reliable Component-based Software Systems.* Artech House Publishers, 2002.
10. E. Motta, J. Dominigue, L. Cabral, and M. Gaspari. IRSII: A Framework and Infrastructure for Semantic Web Services. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 306–318. Springer-Verlag, LNCS 2870, 2003.
11. C. Pahl. An Ontology for Software Component Matching. In *Proc. Fundamental Approaches to Software Engineering FASE'2003.* Springer-Verlag, LNCS Series, 2003.
12. C. Pahl and M. Casey. Ontology Support for Web Service Processes. In *Proc. European Software Engineering Conference and Foundations of Software Engineering ESEC/FSE'03.* ACM Press, 2003.
13. P. Bouquet, L. Serafini, and S. Zanobini. Semantic Coordination: A New Approach and an Application. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 130–145. Springer-Verlag, LNCS 2870, 2003.
14. D.J. Mandell and S.A. McIllraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003*, pages 227–226. Springer-Verlag, LNCS 2870, 2003.
15. R. Zhang, I.B. Arpinar, and B. Aleman-Meza. Automatic Composition of Semantic Web Services. In *Proc. International Conference in Web Services ICWS'2003.* 2003.
16. S. Narayanan and S.A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. World-Wide Web Conference WWW'2002.* 2002.
17. World-Wide Web Conference WWW'2003. *Semantic Web Services Panel.* ACM, 2003.
18. L. Chen, N. Chadbolt, C.A. Goble, F. Tao, S.J. Cox, C. Puleston, and P.R. Smart. Towards a Knowledge-Based Approach to Semantic Service Composition. In D. Fensel, K.P. Sycara, and J. Mylopoulos, editors, *Proc. International Semantic Web Conference ISWC'2003.* Springer-Verlag, LNCS 2870, 2003.
19. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. Semantic Configuration Web Services in the CAWICOMS Project. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.
20. J. Bitcheva, O. Perrin, and C. Godart. Cooperative Process Coordination. In *Proc. International Conference in Web Services ICWS'2003.* 2003.
21. S. Ran. A Framework for Discovering Web Services with Desired Quality of Services Attributes. In *Proc. International Conference in Web Services ICWS'2003.* 2003.
22. M. Paolucci, T. Kawamura, T.R. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In I. Horrocks and J. Hendler, editors, *Proc. First International Semantic Web Conference ISWC 2002*, LNCS 2342, pages 279–291. Springer-Verlag, 2002.

23. K. Sivashanmugan, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. In *Proc. International Conference in Web Services ICWS'2003*. 2003.
24. F. Plasil and S. Visnovsky. Behavior Protocols for Software Components. *ACM Transactions on Software Engineering*, 28(11):1056–1075, 2002.
25. F. Leymann. Web Services Flow Language (WSFL 1.0), 2001. http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
26. C. Szyperski. Component Technology - What, Where, and How? In *Proc. 25th International Conference on Software Engineering ICSE'03*, pages 684–693. 2003.
27. Bertrand Meyer. Applying Design by Contract. *Computer*, pages 40–51, October 1992.
28. I. Horrocks, D. McGuiness, and C. Welty. Digital Libraries and Web-based Information Systems. In F. Baader, D. McGuiness, D. Nardi, and P.P. Schneider, editors, *The Description Logic Handbook*. Cambridge University Press, 2003.
29. R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
30. D. Sangiorgi and D. Walker. *The π-calculus - A Theory of Mobile Processes*. Cambridge University Press, 2001.
31. I. Sommerville. *Software Engineering - 6th Edition*. Addison Wesley, 2001.
32. World Wide Web Consortium. *Web Services Architecture Definition Document*. http://www.w3.org/2002/ws, 2003.
33. A. Moorman Zaremski and J.M. Wing. Specification Matching of Software Components. *ACM Trans. on Software Eng. and Meth.*, 6(4):333–369, 1997.
34. R.J.R. Back and J. von Wright. *The Refinement Calculus: A Systematic Introduction*. Springer-Verlag, 1998.
35. Bertrand Meyer. *Eiffel: the Language*. Prentice Hall, 1992.
36. Dexter Kozen and Jerzy Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 789–840. Elsevier, 1990.
37. C. Li and C. Pahl. Security in the Web Services Framework. In *Proc. International Symposium on Information and Communications Technologies ISICT'03*. 2003.

A Proposal for a Semantic Web Service Description Format

Joachim Peer[1] and Maja Vukovic[2]

[1] Institute for Media and Communications Management, University of St. Gallen, Switzerland
[2] Computer Laboratory, University of Cambridge, United Kingdom

Abstract. Automatic evaluation and consumption of Web services requires a comprehensive semantic information model of a Web service. One example of a language that facilitates capability-driven description of services is OWL-S. However, it imposes two limitations: (1) it offers no native support for the description of certain rule types often needed for service description and (2) it leads to very large service description documents that are difficult to read and write. In this paper, we propose an XML-based markup format that addresses these problems and allows for semantic annotation of Web services of different technical flavors. This work is inspired by OWL-S, however it makes certain design decisions aimed to increase the ease of use of semantic Web service descriptions, by presenting a significantly more compact syntax for service markup and grounding. Furthermore, the proposed description format provides support for non-deterministic service operations.

1 Introduction

Web services are a family of distributed software components that can be exposed and invoked over the Internet. Examples are SOAP based services and XML based REST services. Commonly, interface description languages such as the Web Services Description Language WSDL [1] are used to describe the syntactical interface of a Web service and the message streams it uses to communicate with its clients.

Research in *semantic* Web services aims to develop methods to explicitly describe the semantic and pragmatic aspects of Web services, in addition to the syntactical descriptions already in place. This can be achieved by explicitly describing the logical dependencies between the inputs and outputs of an operation as well as the changes to the world state the execution of a particular operation entails. This way, software agents can distinguish between semantically different operations with equivalent syntactic interfaces. Ultimately, users should be able to describe their needs or goals in a convenient fashion, and automats should be able to identify, invoke, compose and monitor services to accomplish the user's goals.

However, the most advanced description format, OWL-S, has two key usability problems which need to be addressed in order to enable the adoption outside of the academia: OWL-S leads to very large service description documents that are difficult to read and write, while it offers no native support for the description of certain rule types often needed for service description. This paper aims to contribute to the field by providing a description format which addresses those limitations.

The paper is structured as follows. Section 2 discusses related work. Sections 3 and 4 analyze the roles of OWL and RDF in OWL-S respectively, reporting on the strengths and weaknesses of OWL-S. We then propose an approach to overcome the identified limitations of OWL-S. Section 5 provides an overview of our proposed description format. Sect. 6 describes the core component of the markup schema, the functional profile of services. Finally, Sect. 7, concludes and outlines the areas for future work.

2 Related Work and Contribution

The most influential work in the area of semantic Web service description is OWL-S [2]. To the best of our knowledge it is the only published proposal for semantic Web service description that provides a concrete language for service markup. Another emerging language, the Web Service Modeling Language (WSML)[1], is currently under development. At present, only syntactical definitions but no conceptual description of WSML are available. Further related work to be considered is the Web Service Modeling Framework (WSMF) [3], which is dedicated to the definition of viable architectural models for semantic Web services but does not provide any concrete markup language. For these reasons, we will focus our analysis on OWL-S.

OWL-S is an ontology for service description based on the Web Ontology Language (OWL) [4]. The OWL-S ontology consists of the following three parts: a service *profile* for advertising and discovering services; a *process model*, which describes a service's operation in detail; and the *grounding*, which provides a bridge between the abstract semantic description and the concrete technical usage of a service. Individual annotators can use the vocabulary defined by OWL-S to provide semantic annotations of services, and automatic agents may process this information.

However, OWL-S has been reported to be difficult to learn and use [5], which could limit its adoption. In the following two sections, we explore the roots of these problems. In particular, we investigate the consequences of employing OWL and RDF as languages for semantic Web service description and we point out alternative techniques, which are easier to learn and handle.

3 The Role of OWL in OWL-S

All language components of OWL-S and the relationships between them are expressed as OWL concept expressions. OWL is based on description logics [6], a well understood subset of first order predicate logic [7].

[1] http://www.wsmo.org/wsml/

3.1 The Advantages of Using OWL

Description logics allow for efficient reasoning tasks like subsumption checks, to determine the relationship between two given concepts.

In the Web service description domain, this property of description logics is of utmost interest, because service retrieval inherently deals with relationships between concepts. For instance, to judge whether some service operation that requires arguments of type A can be used to process an instance of type B, we need to test whether B represents a sub-concept of A.

Besides checking subsumption of parameters, OWL-S markup allows for such subsumption checks over preconditions and effects as a whole. This approach is similar to earlier work on software component retrieval, e.g. [8], where component compatibility was determined by comparing the pre- and postconditions of their operations.

3.2 The Disadvantages of Using OWL in OWL-S

The problem of using OWL to express preconditions and effects of operations is that it is not sufficiently expressive to capture the operations' semantics. The two key reasons for this are:

1. OWL does not allow for the use of logic variables as is the case in rule languages. While some universally quantified variables may be expressed as OWL properties, free or existentially quantified variables cannot always be expressed in pure OWL.
2. Description logics like OWL cannot, by design, express logical formulas that escape the corset provided by the respective description logic. However, when describing Web services, we frequently require the use of rules that do not exactly fit into that framework, e.g. "if the price is below the credit card limit, the transfer may take place".

Emerging proposals like SWRL [9] and DRS [10] are witnesses to this problem; SWRL and DRS are languages (built on OWL and RDF, respectively) to express rules that cannot be expressed natively in OWL and RDF. However, the semantics of SWRL and DRS are extensions to the semantics of RDF and OWL, which render traditional RDF and OWL reasoning virtually useless and requires specialized reasoners.

A similar problem can be identified regarding processes. While OWL is used to mark up processes, pure OWL cannot capture the semantics of the OWL-S process model; this semantics is given in [11]. This means, however, that pure OWL based reasoning cannot deliver critical reasoning services such as reachability, liveness or deadlock analysis. Again, OWL is used merely as a schema definition language than as a logical concept specification language.

3.3 Towards a More Adequate Use of OWL

We propose to use languages different than OWL to model conditions and effects of operations and to restrict the use of OWL to modeling the relationships between tangible concepts used in Web service domains: (i) Concepts that describe input types of operations (e.g. `CreditCard`), (ii) concepts that describe output types of operations (e.g. `Receipt`), (iii) concepts that describe predicates used in free formulas (e.g. `Debit`), (iv) concepts that describe non-functional properties of a service (e.g. industry = `Tourism`).

This way, we are not affected by the limitations of description logics as outlined above, but we still benefit from its strengths: Using description logics, service annotators can derive their own concepts from standard ontologies while agents could still interpret those concepts to a certain degree. Further, OWL permits to define vocabularies that bridge between different concepts of different ontologies, reducing the heterogeneity of the concepts involved in service descriptions.

To sum up, we embrace description logics as a means for "terminology / vocabulary management" but we do not use it for modeling conditions and effects of operations.

4 The Role of RDF in OWL-S

OWL is built on top of the Resource Description Format (RDF) [12]; which describes networks of data as sets of binary propositions *P(S, O)*, so-called RDF triples. Each such triple comprises a "statement", connecting a subject S to an object O via a relation ("property") P. All components P, S, O of an RDF statement are identified by a URI [13].

All OWL documents, including OWL-S documents, are represented as sets of RDF triples. In addition, RDF may be used freely to provide additional information about OWL-S annotations.

4.1 The Advantages of Using RDF

Viewing RDF triples as statements about entities defined by URIs is an attractive approach, as it allows everyone to make statements about everything that has a URI. This allows us to collaboratively create networks of statements, which leads to the idea of a "semantic Web", as proposed by [14].

4.2 The Disadvantages of Using RDF in OWL-S

The elegance of RDF comes at a price. Many types of data cannot be directly expressed solely by means of binary relations (triples). Constructs that span over more then two data items and more than one property are not very intuitively represented in RDF; for instance it is difficult to express the fact that some article is "the 32nd item on the bill of some customer" in RDF.

Further, RDF Schema does not provide for validation tests as strict as XML Schema or XML DTD validation. For instance, in XML it is possible to exclude the nesting of certain XML elements, while RDF Schema cannot be used to prevent unwanted RDF statements. Furthermore, RDF documents (serialized via RDF/XML, N3) are less readable than pure XML documents, especially when ordered collections of objects or reification are involved (albeit to some degree, this is a matter of personal taste).

These usability problems seem to be reflected by the slow adoption of RDF. RDF and RDF Schema are available since 1998 but in contrast to XML, they did never enjoy a widespread popularity among mainstream developers or data engineers.

4.3 Towards a More Adequate Use of RDF

As described above, key advantage of RDF is that it allows to make "statements" about arbitrary things that are denoted by a URI. RDF can be used to freely publish additional statements about a service markup to express information not foreseen by OWL-S.

We argue that the service markup annotated via RDF need not be an RDF document itself. Instead, an XPointer framework can be employed to create RDF statements on XML documents. Using XPointer, every node of an XML document can be uniquely addressed using "fragment identifiers". For instance, the URI of a certain WSDL operation can be uniquely identified by a pointer like: http://example.org/TicketAgent.wsdl20#operation(TicketAgent/reserveFlight)

Since fragment identifiers are supported by RDF[2], our XML based service specifications and annotations can still be interweaved with the RDF based semantic Web; everybody can publish a set of RDF triples with subjects, objects or properties pointing to fragments of our service descriptions. We argue that an XML based service markup is just as compatible to the semantic Web as any RDF based document is. But at the same time, our markup enjoys the ease of use of XML, as we will show in the sections to follow.

5 Proposed Semantic Markup

In the light of these statements, we propose a semantic markup format that is inspired by OWL-S but differs from it in several aspects. Firstly, our markup uses OWL only in the restricted way outlined in Sect. 3.3; secondly, it uses XML instead of RDF, for reasons given in Sect. 4.2.

Our model of semantic service annotation is presented in Fig. 1. Following this model, a semantic service annotation consists of two parts: a *functional* profile, which describes the semantics of the operations, and a *non-functional* profile which allows for semantic annotation of the service as a whole (e.g. information about the provider, geographic availability, etc.).

[2] cf. http://www.w3.org/TR/rdf-concepts#section-fragID

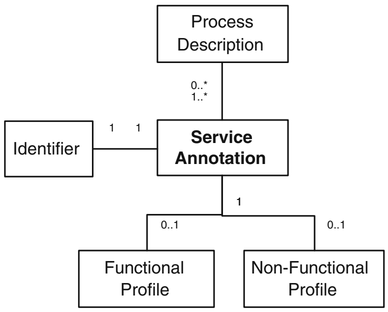


Fig. 1. Compontents of semantic service annotation

The semantic description of a service may be used for the description of *processes*, which are patterns of interactions between the user and a number of services. Each defined process may refer to a number of services, but not every service description is required to be part of such a process description.

As the reader may note, there is no explicit service grounding in the model, in contrast to the OWL-S model. As we will see later, the service grounding is captured by the functional profile.

We see semantic service description as an attachment to the syntactic service description; an agent needs *semantic* service description to determine the usefulness of a service for a particular task and it needs *syntactic* service description to successfully interact with the service. To provide a seamless integration of syntactic and semantic service information, we impose the following requirements:

– on service description: There must exist syntactic information that tells agents how to access a service (e.g. a WSDL document)
– on operation description: There must exist syntactic information that tells agents how to access a service's operation (e.g. an `operation` element in WSDL)
– on message description: The input and output messages of the service's operations must be based on a structured format and the definition (e.g. an XML Schema document) of those message formats must be accessible.

The XML fragment below shows the root element of a semantic service annotation using our proposed markup:

```
<service-annotation xmlns:wsdl="http://schemata.org/sws/wsdl"
 wsdl:url="http://my.com/service.xml" wsdl:name="MyService">
  <functional-profile>
    <!-- description of operations -->
  </functional-profile>
  <nonfunctional-profile>
    <!-- description of non-functional aspects  -->
  </nonfunctional-profile>
</service-annotation>
```

Web services of different technical flavors (WSDL/SOAP, REST, etc.) differ in the way the services, operations and message descriptions discussed above are provided. To achieve interoperability of our markup with those different technical implementations of Web services, we need to provide extensibility mechanisms. Our proposed solution is to use qualified attributes to provide the technology-dependent information. In the case of WSDL, the agent will have to retrieve the information from the WSDL document stored in the URL specified by the `wsdl:url` attribute, to identify the syntactic requirements of the service whose name is given by the `wsdl:name` attribute. Of course, the agent must be aware of the meaning of the technology specific information to make use of it.

In the following, we will present a markup format to express the functional aspects of services.

6 The Functional Profile

The functional profile provides information about the semantics of a service's operations. Agents may use those annotations to analyze the conditions and consequences of invoking an operation, and to determine if such an invocation would be useful to them.

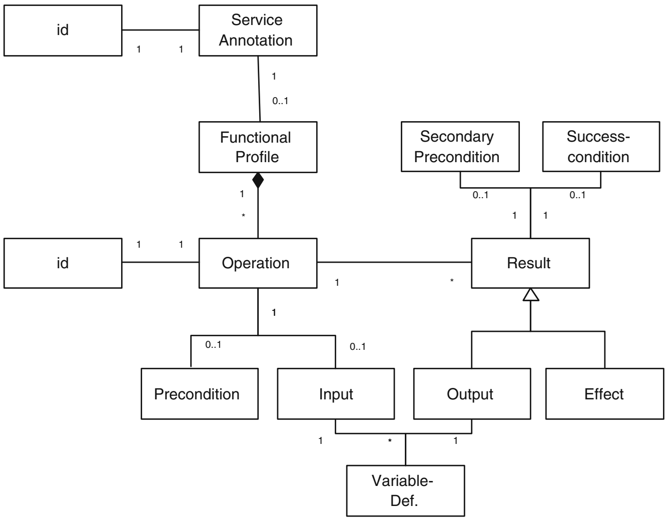


Fig. 2. The functional profile of a service

6.1 Marking Up Operations

Each service operation that should be made available to semantic Web agents needs to be marked up, using an `<op-def>` element.

Just like service annotations, operation annotations also need to be linked to technology specific information. This is necessary because different Web service technologies use different ways to identify and describe the syntactic properties of operations: for example, with WSDL based descriptions, an operation is characterized by a port type and an operation name (in addition to the WSDL URI and service name); with REST based Web services, however, an operation is characterized by an HTTP command combined with a URI that matches a certain pattern (e.g. PUT http://test.com/addToCart/*)

Again we use prefixed XML attributes to capture the technology specific information. Below we show an example of an annotation of a WSDL based service operation, which is identified by its name and port type:

```
<op-def wsdl:name="ActivateUser" wsdl:portType="RSAFuncsSoapPT">
  <!-- information about operation's semantics -->
</op-def>
```

The language constructs that can be used to capture the semantics of an operation, as illustrated by Fig. 2 are:

– an optional element `input` to define the data pieces that are expected as inputs. The input is defined as a set of variable definitions. Each variable definition connects a logical variable to concrete elements of a service's input stream. Agents may use this information to correctly construct the messages they send to the service.
– an optional element `precondition` to describe constraints that need to be satisfied by both the inputs and by the world state (as represented by the agent's knowledge base) in order to successfully execute the service operation.
– a number of (possible) service results. We distinguish two (complementary) types of results: `outputs`, which describe the pieces of data the service will return, and `effects`, which describe the expected state of the world (including constraints on the expected outputs) after the operation is executed.
– Every result may have a `secondary-precondition` that is required in addition to the operation's precondition. This allows to model conditional outputs and conditional effects.
– Furthermore, a `success-condition` may be defined for each effect. In contrast to the other conditions, this condition does not need to be true prior to service execution, but it has to be true afterwards. This allows for dealing with non-deterministic services, assuming execution monitoring on the agent's side.

We provide more details on these constructs. First, we describe the formalism used to model these constructs and present a plain Lisp-like notation. We use this notation throughout this paper for sake of brevity. However, in Sect. 6.2, we introduce alternative XML based representations of formulas and conditions.

Conceptual Model of Operation Descriptions. As discussed in Sect. 3, semantic service annotation does not fit into the tight framework of description logics, even if they are as expressive as OWL. Instead, we often require freely defined logical formulae that include logic variables, explicit quantifiers etc. On the other hand, the consequences of too liberal use of logical constructs are well known (i.e. computational intractability).

A viable compromise that seems neither too restrictive nor too expressive is the Planning Domain Definition Language (PDDL) [15]. PDDL is used by the AI planning community to describe, among other things, the semantics of operations (also called operators, action schemata) of planning domains. PDDL allows to model the preconditions and (conditional) effects of actions using a restricted subset of first order logic. The BNF definition of the PDDL version we are using can be looked up in [15]; examples of PDDL's simplifications over pure first order logic are:

- PDDL partitions an action's dynamic aspects into `precondition` and `effect` sections. This allows to better balance the expressivity of the descriptions, because reasoning over complex effects usually is more demanding for agents than reasoning over preconditions.
- In fact, PDDL allows existential quantifiers in preconditions but forbids them in effects.
- PDDL prohibits the use of functions (of arity > 0), and therefore nested functions are not supported as well.

As stated above, we will first present a plain Lisp-like representation, directly taken from PDDL. In addition, we support namespace prefixes for predicates and constants, as proposed for Web-PDDL [16]. This is helpful to connect some of the involved predicates and constants to semantic Web ontologies, which we support as discussed in Sect. 3.3. Just like in XML, the prefixes used must be defined in the namespace declaration of one of the ancestor elements of the respective description. If no prefix is given, the empty namespace is assumed.

Inputs to Operations. The syntax of inputs is defined by the syntactic service description, e.g. the WSDL message definition of the operation. However, to connect certain relevant data pieces to the constraints expressed by preconditions and effects, additional markup is needed.

```
<input>
   <var name="?to" wsdl:part="ToAddress"/>
   <var name="?from" wsdl:part="FromAddress"/>
   <var name="?subject" wsdl:part="Content" wsdl:path="Subject"/>
   <var name="?msg" wsdl:part="Content" wsdl:path="Message"/>
</input>
```

As shown in the example above, a variable is introduced by an element `var` and its name is defined by an attribute `name`. In case the underlying description format is WSDL, we use an attribute `wsdl:part` to connect the variable to the correct WSDL message part and we use an optional attribute `wsdl:path` to specify the XPATH of the referenced piece of data within the given message part.

This way, pieces of data sent from and to the service can be connected to variables used in the `precondition`, `secondary- precondition` and `effect` formulas (cf. the sections below) in order to put semantic constraints on these data tokens. The connection between logical variables in conditions and effect formulas on the one hand and the inputs (and outputs) on the other hand is illustrated in Fig. 3.

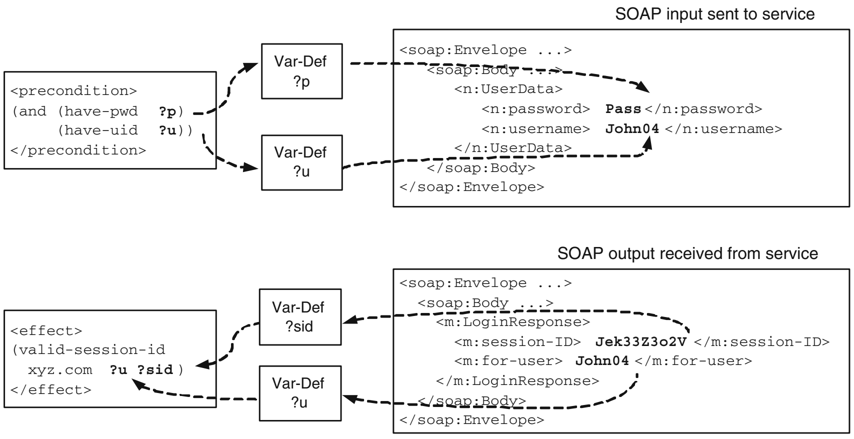


Fig. 3. Variable definitions connect semantics and concrete syntax

This information is conceptually roughly equivalent to the OWL-S grounding information, and can be used by the agents to properly construct outgoing messages when invoking the service and to properly interpret the values of incoming messages.

Outputs of Operations. In analogy to inputs, `output` definitions allow to specify the data pieces that may be represented by the logical variables used in the service markup.

In the example below, three data pieces are identified and connected to the variables `?x`, `?c` and `?p`, referring to the `id`, `color` and `price` of some `Item`.

```
<output>
   <var name="?x" wsdl:part="result" wsdl:path="Item/id"/>
   <var name="?c" wsdl:part="result" wsdl:path="Item/color"/>
   <var name="?p" wsdl:part="result" wsdl:path="Item/price"/>
</output>
```

The semantics of outputs is related to sensing actions as discussed in the AI planning domain, characterized e.g. by *learnable terms* [17] or as *observations* in NPDDL [18]; the outputs of services can be used to add information to the agent's knowledge base. We will discuss this aspect below in the context of effect formulas and in Sect. 6.2.

Preconditions of Operations. Just like in PDDL, we define a precondition as a function free first order logical sentence[3]. A precondition describes the requirements that must hold in order to achieve any of the operation's results. If a precondition is not fulfilled and an operation is still called, the results are undefined.

The following example shows a precondition that states that the operation can only be called on a Sunday and if the caller has a membership of `MyCompany` (the namespace prefixes `n` and `m` are referring to some external vocabularies):

```
<precondition xmlns:n="http://biz.org/voc"
              xmlns:m="http://my.com/def">
  (and (n:day-of-week Sunday)
       (n:have-membership m:MyCompany))
</precondition>
```

Effects of Operations. As in PDDL, we define an effect formula as a function free first order logical sentence which may be universally (but not existentially) quantified. An agent that calls an operation with a valid precondition, can expect that the effect formula will evaluate to true in the world state after the operation is executed, as long as no success condition exists (cf. below) and as long as no error occurs.

For instance, to describe the effect of an operation that delivers an SMS message to a cell phone number and charges one credit point from the user's account, the following annotation could be created:

```
<effect>
  (and (sms-sent ?nr ?msg)
       (debit ?account 1))
</effect>
```

To explicitly describe the effect an operation has on the *knowledge* of an agent (by sending output data), we use a predicate `know`. As an example, consider the following markup of an operation that returns the price of a commodity good:

```
<op-def name="getPrice" wsdl:portType="MegashopService">
  <input>
    <var name="?item"/>
    <var name="?ean" wsdl:part="ean"/>
  </input>
```

[3] A sentence is formula that has no free (unquantified) variables.

```
  <precondition>(property ?item s:ean-nr ?ean)</precondition>
  <output>
    <var name="?p" wsdl:part="getPriceReturn"/>
  </output>
  <effect>(know (property ?item s:price ?p))</effect>
</op-def>
```

The effect of the operation is that the price `?p` of the requested item `?item` gets known to the agent. In Sect. 6.2, we will present a more concise syntactic variant to represent this semantics.

Secondary Preconditions of Results. While preconditions as presented above in Sect. 6.1 describe conditions that are required for an operation as a whole (i.e. for all of its possible results), secondary preconditions describe conditions that is required just for a specific output or effect.

For example, take an operation that has a precondition P and two results A and B, whereby A has a secondary precondition SP_A and B has no such secondary precondition. Under this circumstances the result A can only occur if $(P \wedge SP_A)$ holds, whereas B does only require P to hold.

Of course, the separation of conditions into preconditions and secondary preconditions is only syntactic sugar. The precondition element of an operation could be removed by adding it as a conjunction to all the secondary preconditions of the operation's results. However, this would create redundant markup, which we usually want to avoid.

Success Conditions of Results. The success of a Web service operation may be undetermined until the execution is actually over. For instance, an operation that sends e-mail messages may return an error report in case the message could not be delivered.

To provide agents with a tool to cope with this kind of non-deterministic results, we provide a `success-condition` construct that is attached to an operation's result. Like the other conditions described above, a success-condition is represented by a PDDL compliant first order logical sentence.

Generally, we can afford expressive and semantically opaque operations here, because success conditions only need to be evaluated *after* a service operation, when relevant variables are already bound to concrete values, which allows for efficient evaluation of such expressions.

It should be clearly distinguished between the concept of success conditions described in this section and the concept of preconditions and secondary preconditions described earlier above: A (secondary) precondition states that an effect occurs only if its condition is true prior to service execution. In contrast, a *success condition* does not need to be true *before* service execution, but it must be true *after* service execution, to allow the effect to occur legally.

6.2 Refinement of the Syntax

Enriching Inputs and Outputs. In the paragraphs above, precondition formulas have been used to characterize and constrain the data pieces defined as inputs and effect formulas have been used to characterize and constrain the outputs.

An important part of these constraints is *type information*, i.e. information that tells the agents how the data pieces defined align with semantic Web ontologies; in particular, we may want to define (i) the type of a data piece `?i`, e.g. `(class someClass ?i)` and (ii) that the data piece ?o is a property *propertyName* of some other object, e.g. `(property ?otherObject propertyName ?o)`.

Since these type constraints follow a simple schema which basically consists only of two types of predicates (those that specify class membership and those that specify property relationships), we can provide syntactical short-cuts to capture this information. To this end, we introduce the following additional XML attributes:

- `owl:class` is used to define the OWL class membership of the specified data token.
- `owl:isProperty` and `owl:of` are used to specify that the data token represents a property of some other object.

As an example, let us visit the example from above again, now with the syntactic shortcuts applied; as we shall see, the PDDL formulas have disappeared and their semantics are now encoded in `<var>` elements in the input and output sections:

```
<op-def name="getPrice" wsdl:portType="MegashopService">
  <input>
    <var name="?item" owl:class="s:Item"/>
    <var name="?ean" wsdl:part="ean"
          owl:isProperty="s:ean-nr" owl:of="?item"/>
  </input>
  <output>
    <var name="?p" wsdl:part="getPriceReturn"
          owl:isProperty="s:price" owl:of="?item"/>
  </output>
</op-def>
```

Note that this syntax not only spares us from dealing with numerous `class` and `property` predicates, it also spares us from explicitly expressing knowledge effects as expressed by `(know (property ?item s:price ?p))`.

This description schema can be easily handled using XML editors. By providing a certain modeling guideline, it also helps reducing the risk of too heterogenous or even erroneous descriptions. Conditions that do escape this description scheme, however, may still be expressed as arbitrary PDDL formulae in `precondition` and `effect` elements. In the next section, we show how these PDDL formulae can be expressed in XML.

Alternative Syntax for Formulas and Conditions. Throughout the last sections we presented formulas and conditions in Lisp-like syntax used by PDDL (and Web-PDDL). The Lisp-like syntax can be easily crafted using an arbitrary text editor and uses very little space. However, an alternative XML based markup for formulas and conditions is useful for several reasons: firstly, we get validation support from using an XML Schema, secondly it is easier to create XML parsers than to create a Lisp-style formula parser. This should ease the development of compatible semantic Web agents. We use Jonathan Gough's work on XPDDL[4], an XML based serialization for PDDL 2.1. To get an impression of the look and feel of this syntax, consider the following XML based version of the formula presented above in the section on effects:

```
<formula>
  <and>
    <predicate id="sms-sent">
      <parameter id="?nr"/><parameter id="?msg"/>
    </predicate>
    <predicate id="debit">
      <parameter id="?account"/><parameter id="1"/>
    </predicate>
  </and>
</formula>
```

While XPDDL based descriptions are much larger than plain PDDL descriptions, this syntax still consumes considerable less space than the RDF based SWRL and DRS languages used by OWL-S.

7 Conclusion

To overcome the usability limitations imposed by OWL-S, we proposed an XML based markup format for the semantic annotation of Web services. In this paper, we presented the description of operations and their semantics as given by preconditions, results and success conditions. In addition, we illustrated how to bind logical variables to concrete pieces of input and output streams. An XML schema of the proposed format is available online[5], along with an extended version of this paper, illustrating examples and a more detailed comparison with OWL-S.

Open issues for future work are theoretical and empirical elaboration of the complexity and tractability of the language and the embedding of service descriptions in process descriptions.

Acknowledgement. We would like to thank the anonymous reviewers for their valuable suggestions. This research has been funded by the Swiss National Science Foundation under contract 200021-104009 and by the European

[4] http://www.cis.strath.ac.uk/ jg/
[5] http://elektra.mcm.unisg.ch/pbwsc/documents

Commission and the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779. Maja Vukovic is supported by IBM Zurich Research Laboratory.

References

1. W3C: Web Services Description Language (WSDL) version 1.2, http://www.w3.org/tr/2003/wd-wsdl12-20030303/ (2002)
2. The OWL-S Coalition: OWL Web Services V1.0, http://www.daml.org/services/owl-s/1.0/ (2004)
3. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications **1** (2002)
4. McGuinness, D.L., van Harmelen, F.: Feature synopsis for OWL Lite and OWL, http://www.w3.org/tr/owl-features/ (2001)
5. Sabou, M., Richards, D., van Splunter, S.: An experience report on using DAML-S. In: The Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web. (2003)
6. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in description logics. In Brewka, G., ed.: Principles of Knowledge Representation. CSLI Publications, Stanford, California (1996) 191–236
7. Copi, I.M., Cohen, C.: Introduction To Logic. 11th edn. Dark Alley (2001)
8. Fischer, B.: Deduction based software component retrieval (Phd Theses, University of Passau) (2001)
9. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language - Combining OWL and RuleML, http://www.daml.org/rules/proposal/ (2004)
10. McDermott, D.: DRS: A Set of Conventions for Representing Logical Languages in RDF, http://www.daml.org/services/owl-s/1.0/drsguide.pdf (2004)
11. Ankolekar, A., Huch, F., Sycara, K.: Concurrent Execution Semantics of DAML-S with Subtypes. In Horrocks, I., Hendler, J., eds.: Proceedings of The First International Semantic Web Conference (ISWC). Number 2342 in Lecture Notes in Computer Science, Springer-Verlag (2002) 318–332
12. W3C: Resource Description Framework (RDF), http://www.w3.org/rdf/ (2002)
13. Berners-Lee, T., Fielding, R., Irvine, U., Masinter, L., Corporation, X.: Uniform Resource Identifiers (URI): Generic Syntax, http://www.ietf.org/rfc/rfc2396.tx (1998)
14. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American, May '01 (2001)
15. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL– the planning domain definition language. In: http://citeseer.nj.nec.com/ghallab98pddl.html. (1998)
16. McDermott, D., Dou, D.: Representing Disjunction and Quantifiers in RDF. In: Proceedings of the First International Semantic Web Conference, Springer Verlag (2002)
17. McDermott, D.: Estimated-Regression Planning for Interactions with Web Services. In: Proc. of the AI Planning Systems Conference '02. (2002)
18. Bertoli, P., Cimatti, A., Lago, U.D., Pistore, M.: Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In: ICAPS'03, Workshop on PDDL. (2003)

Author Index